From Demand-Aware Networks (DANs) to Self-Adjusting Networks (SANs)

Stefan Schmid et al., most importantly: Chen Avin

GIF
German-Israeli Foundation
for Scientific Research
and Development

From Demand-Aware Networks (DANs) to Self-Adjusting Networks (SANs)

Stefan Schmid et al., most importantly: Chen Avin

New in Austria, looking for collaborations etc. ☺

GIF
German-Israeli Foundation
for Scientific Research
and Development

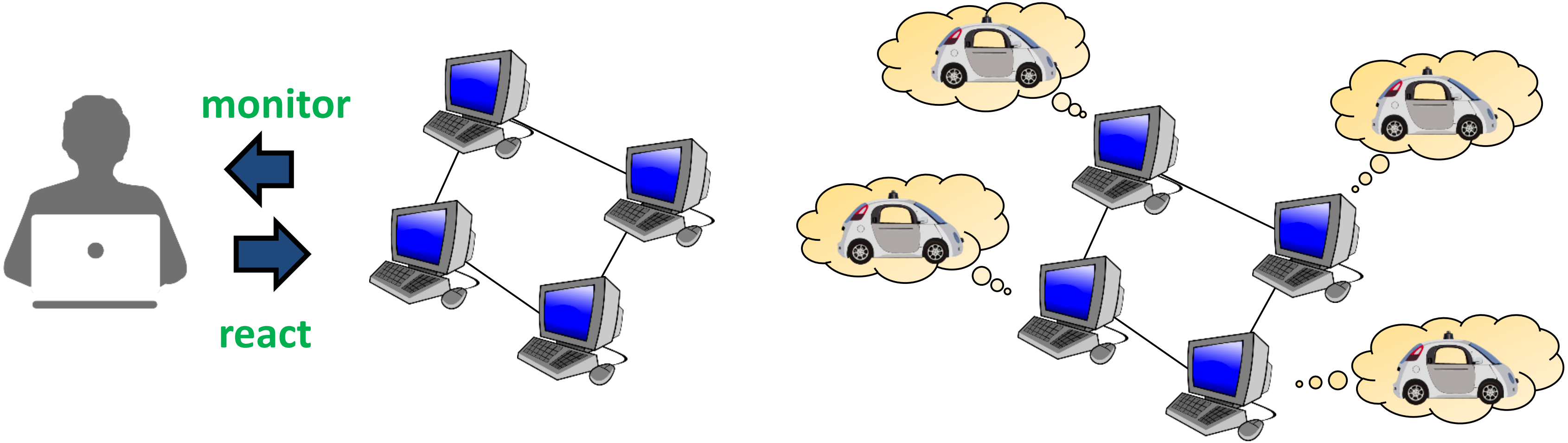
Nice to meet you!

A Brief Overview

- Vision and mission: Make networked systems **self-***
  - Self-repairing
  - Self-stabilizing
  - Self-adjusting

- Using different methodologies
  - **Algorithms** and analysis (LPs, online/approx. algorithms, etc.)
  - **Machine-learning** (data-driven and “self-driving” networks)
  - **Formal methods** (e.g., automata theory and synthesis)

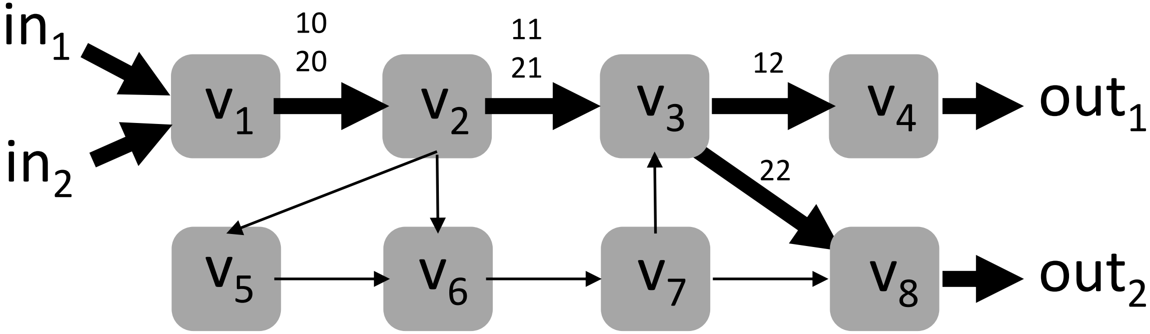
A Brief Overview



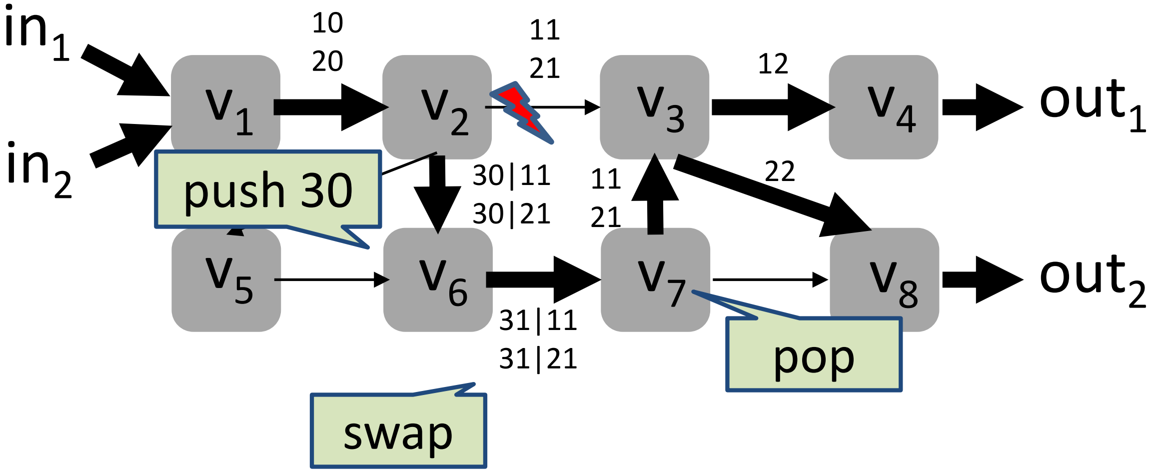
monitor

react

Example: Fast Reroute in MPLS Networks

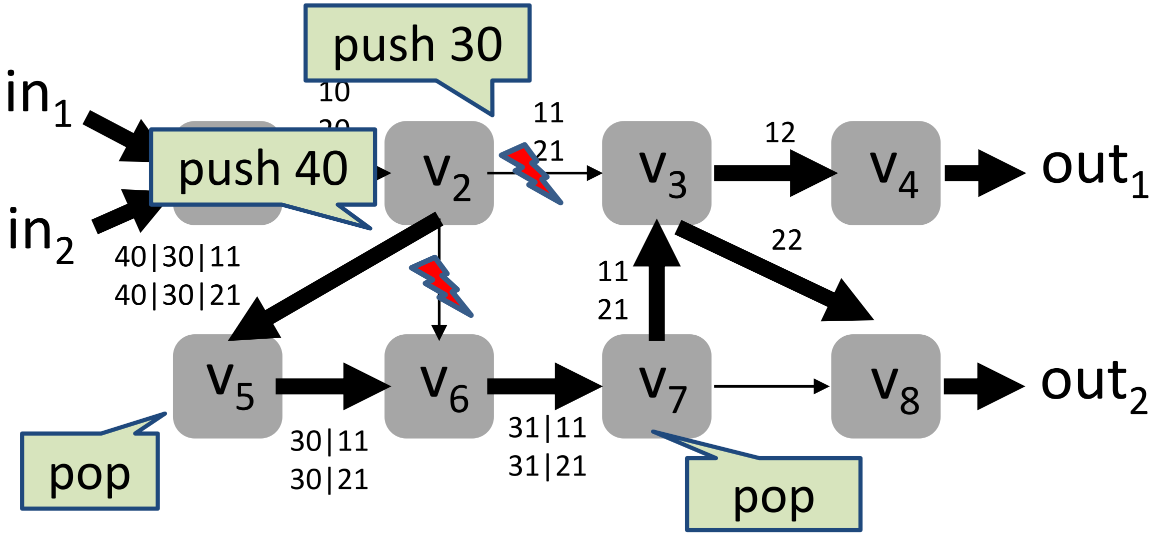


Original Routing



One failure: push 30: route around (v$_2$,v$_3$)

2 Failures

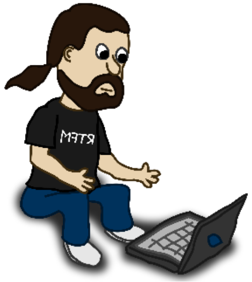


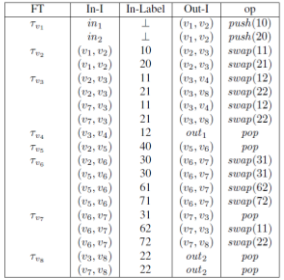
Two failures:

first push 30: route around (v_2,v_3)

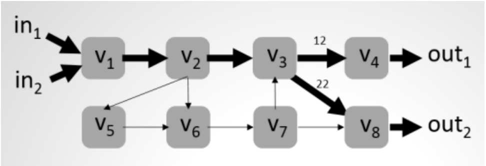
Recursively push 40: route around (v_2,v_6)

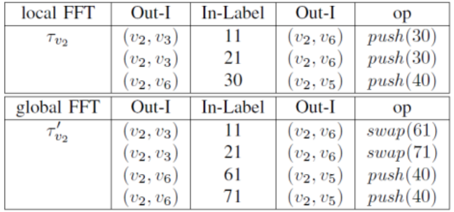
Polynomial-Time What-if Analysis





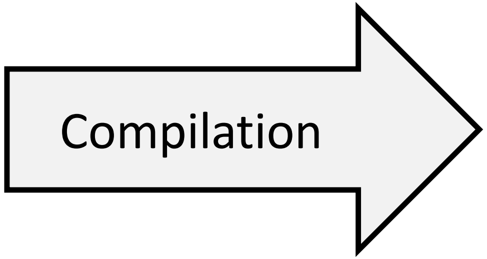
FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\bot$	(v_1, v_2)	$push(10)$
	in_2	$\bot$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop





local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$
global FFT	**Out-I**	**In-Label**	**Out-I**	**op**
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

MPLS configurations, etc.



Compilation

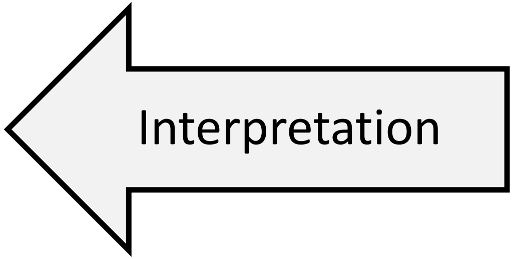
$pX \Rightarrow qXX$

$pX \Rightarrow qYX$

$qY \Rightarrow rYY$

$rY \Rightarrow r$

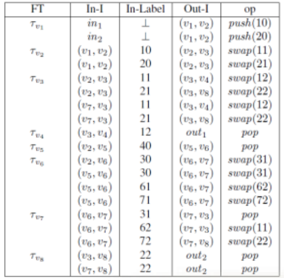
$rX \Rightarrow pX$



Interpretation

Prefix Rewriting System
and Push-Down
Automata Theory

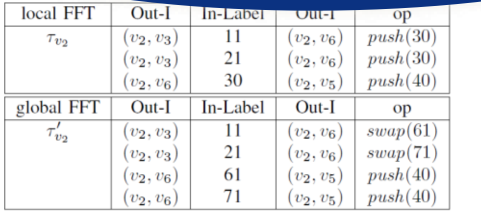
Polynomial-Time What-if Analysis



FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop

Polynomial-time (arbitrary failures)!

Büchi



local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$

global FFT	Out-I	In-Label	Out-I	op
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

MPLS configurations, etc.

Compilation

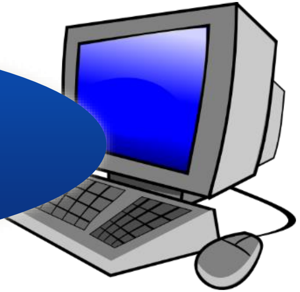
$pX \Rightarrow qXX$

$pX \Rightarrow qYX$

$qY \Rightarrow rYY$

$rY \Rightarrow r$

$rX \Rightarrow pX$

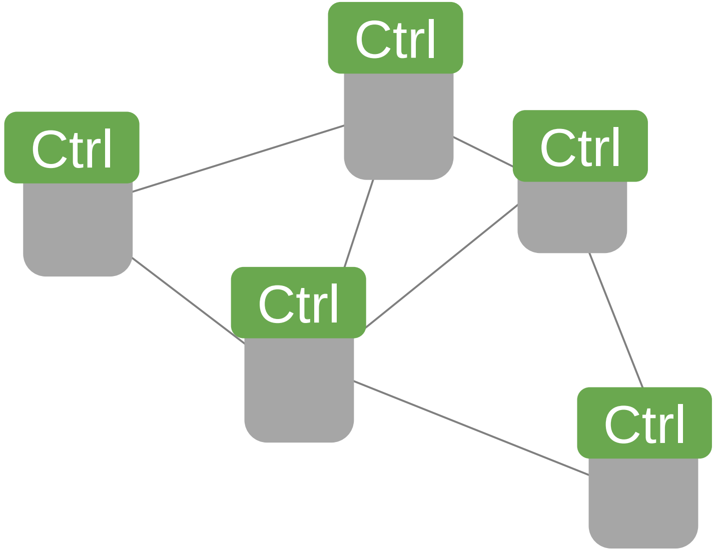


Interpretation

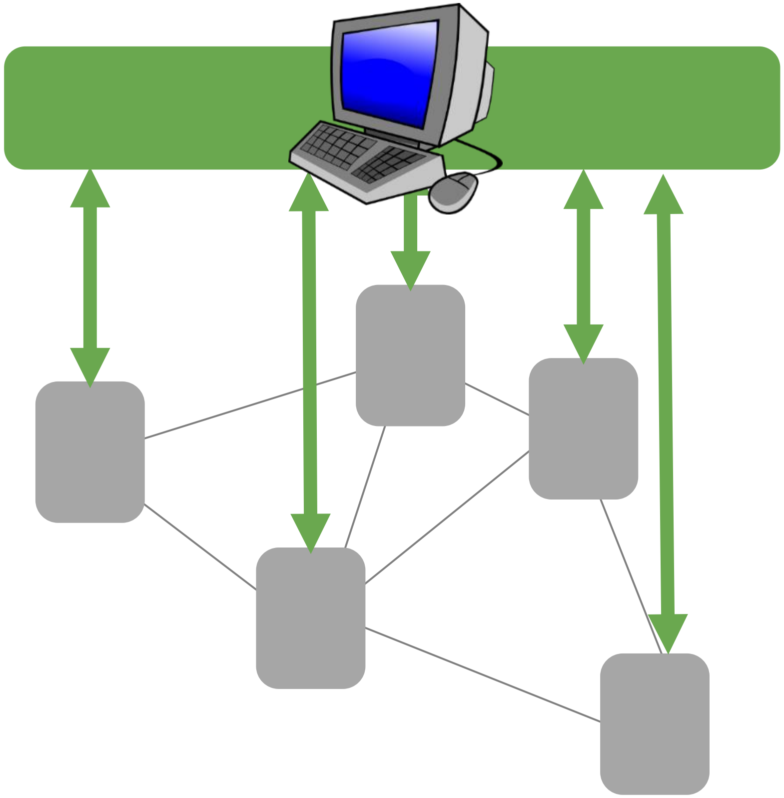
Prefix Rewriting System
and Push-Down
Automata Theory

IEEE INFOCOM 2018

Software-Defined Networks



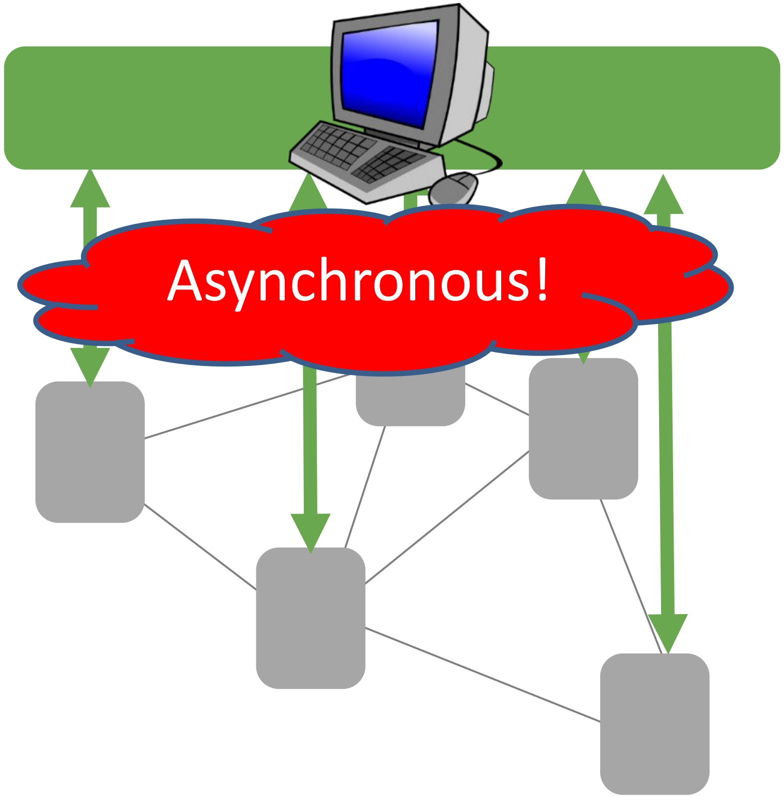
Traditional networks: algorithms and functionality fixed, blackbox



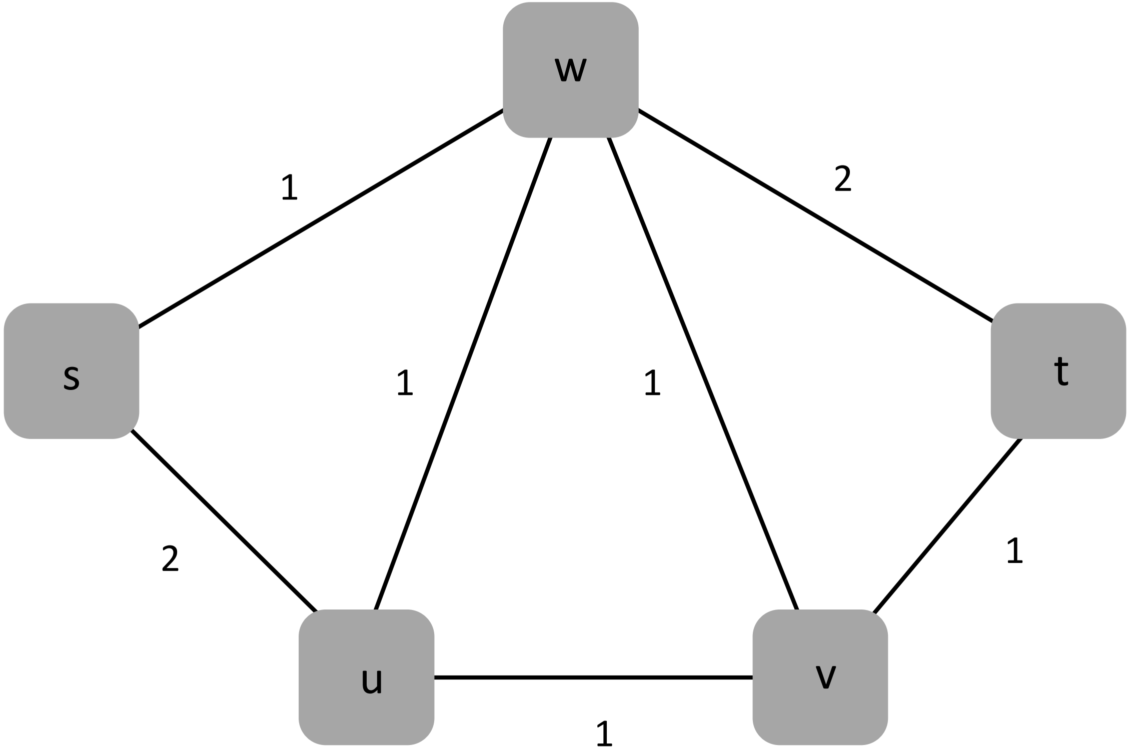
Software-defined networks: bring your own algorithm, match-action (formally verify)

Software-Defined Networks

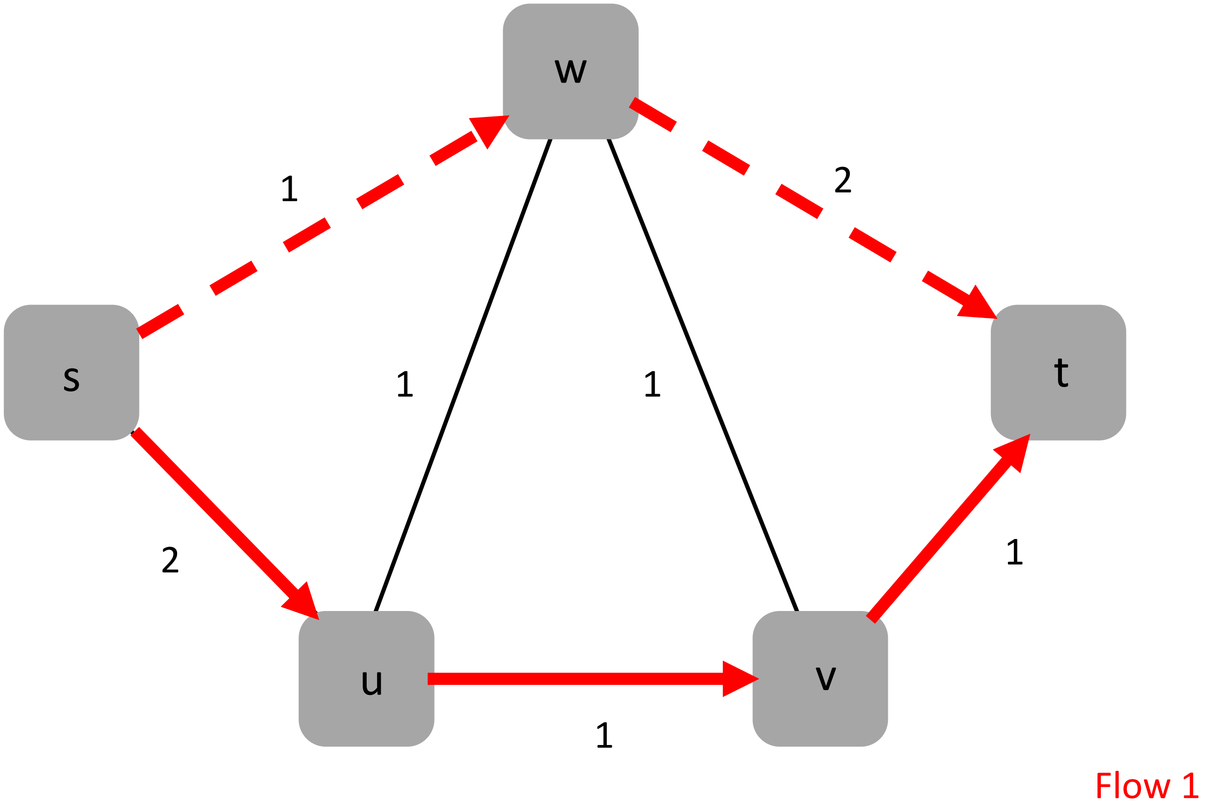
- Software-defined network and network virtualization: networks become **software** and **open**
  - “the Linux of networking”
- Challenges:
  - More **expressive forwarding**: match-action on Layer-2 to Layer-4
  - Complex **verification**



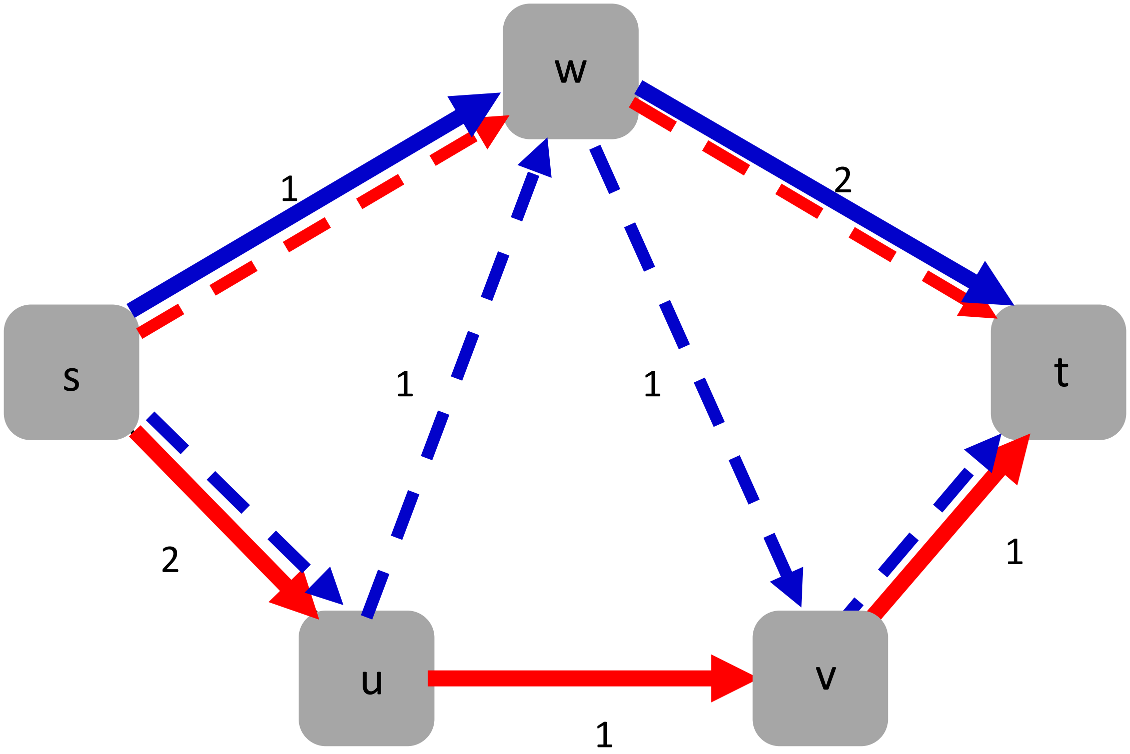
Consistent Network Updates



Consistent Network Updates



Consistent Network Updates

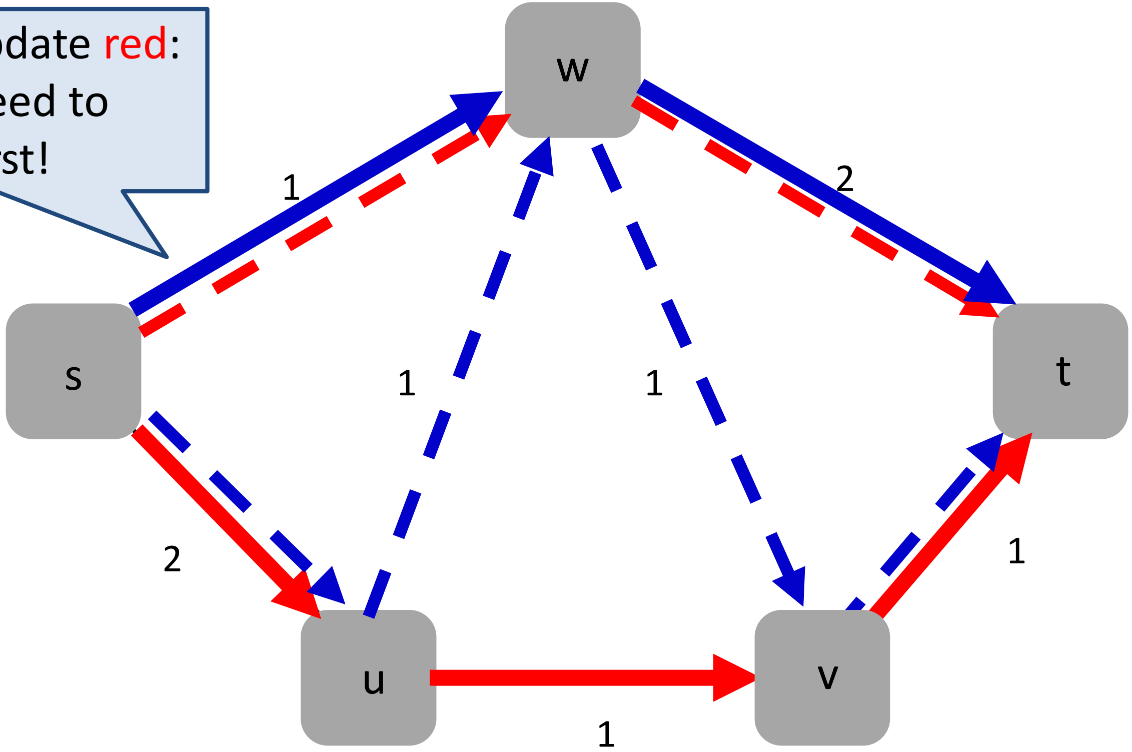


Flow 1

Flow 2

Can you find an update schedule?

Consistent Network Updates

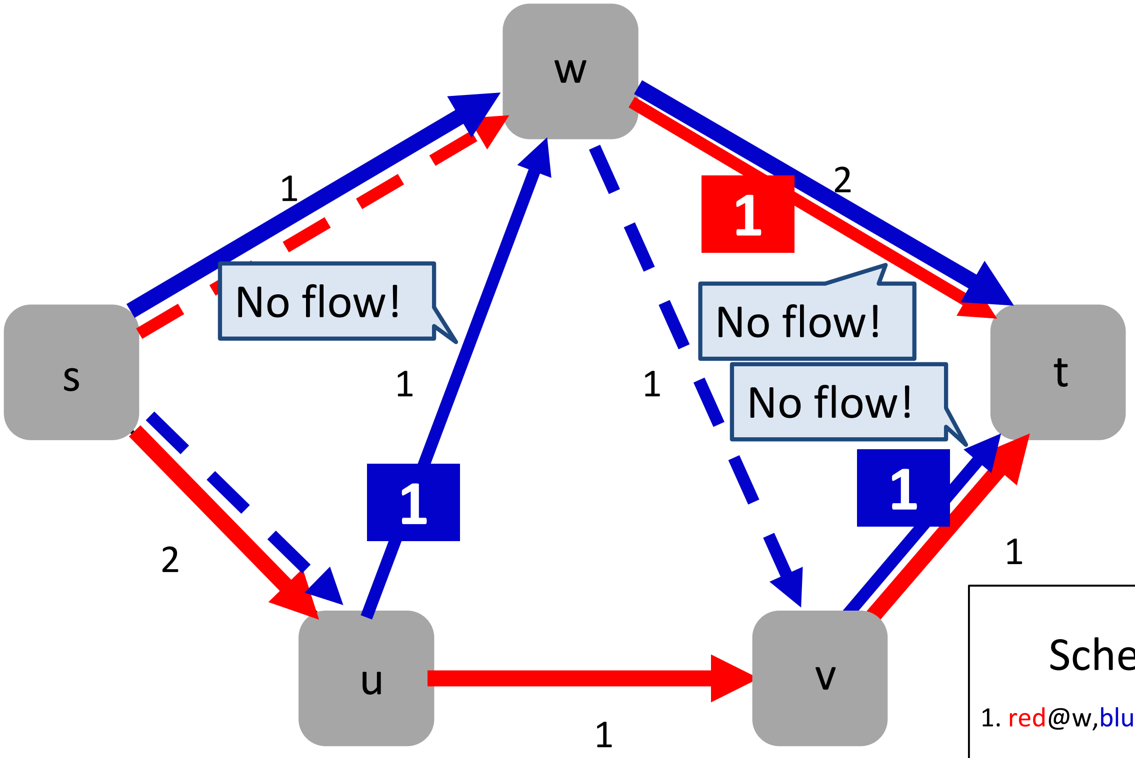


e.g., cannot update red: congestion! Need to update blue first!

Can you find an update schedule?

Flow 1
Flow 2

Consistent Network Updates

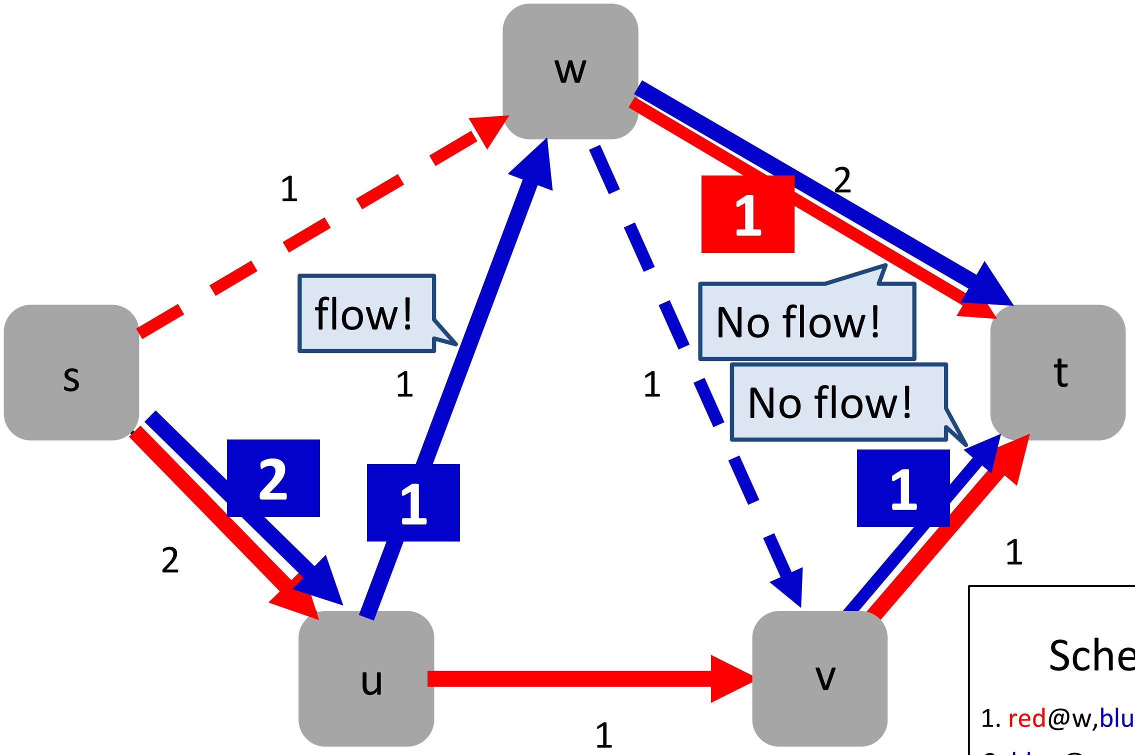


Schedule:

1. red@w,blue@u,blue@v

Round 1: prepare

Consistent Network Updates

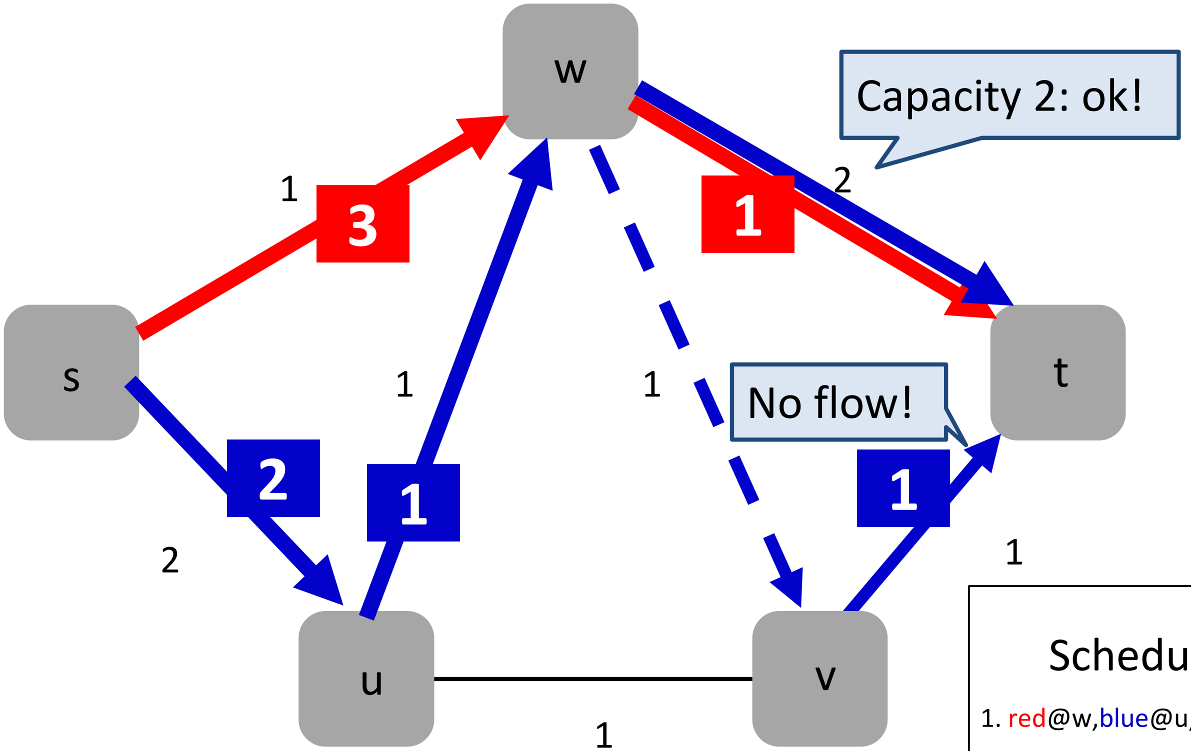


Schedule:

1. red@w,blue@u,blue@v
2. blue@s

Round 2

Consistent Network Updates



Capacity 2: ok!

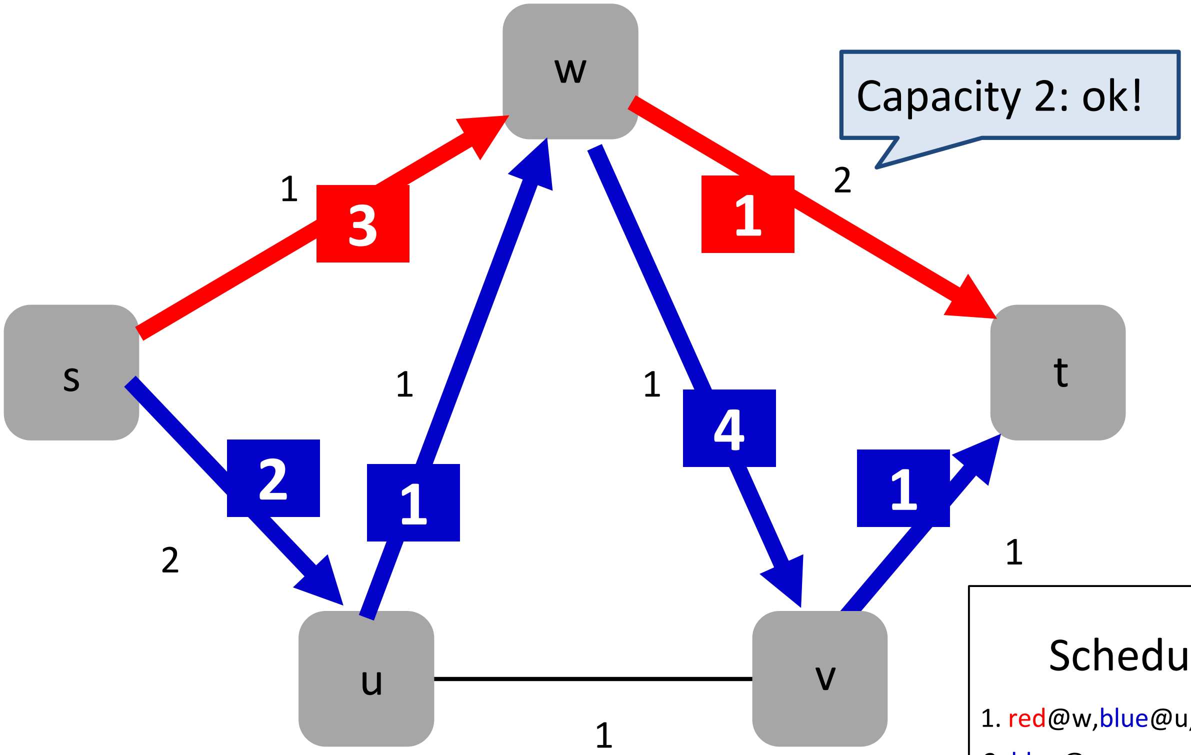
No flow!

Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s

Round 3

Consistent Network Updates



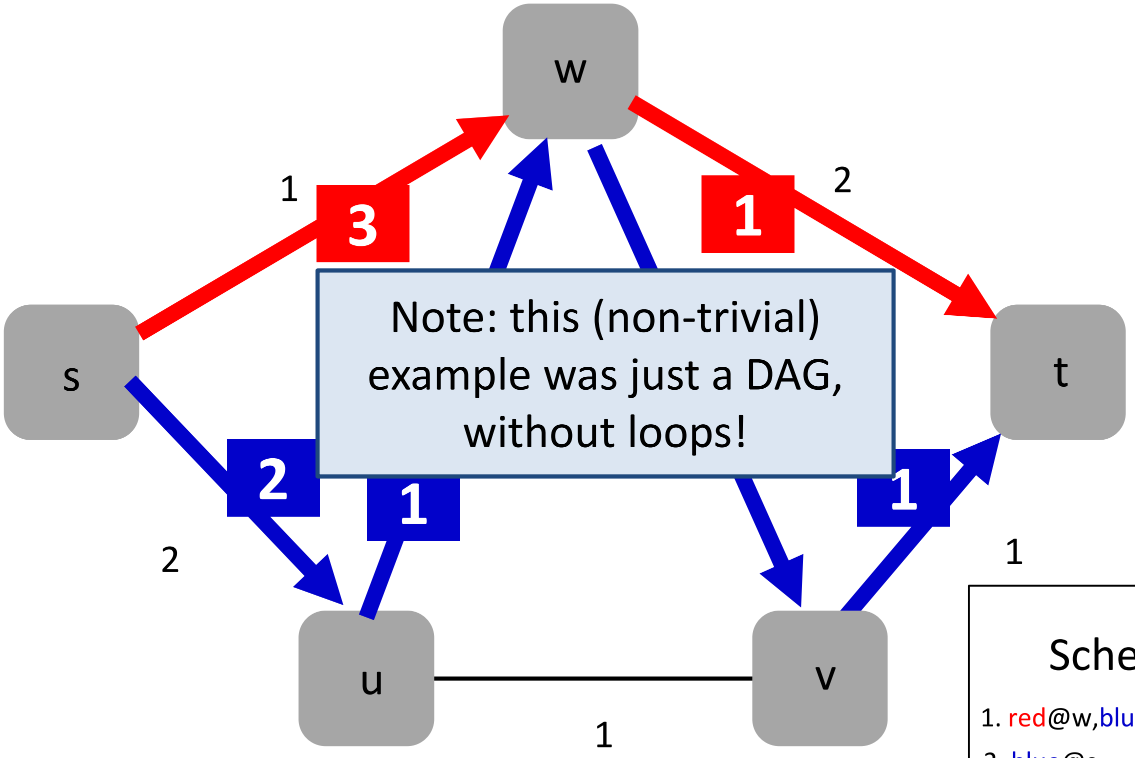
Capacity 2: ok!

Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s
4. blue@w

Round 4

Consistent Network Updates



Note: this (non-trivial) example was just a DAG, without loops!

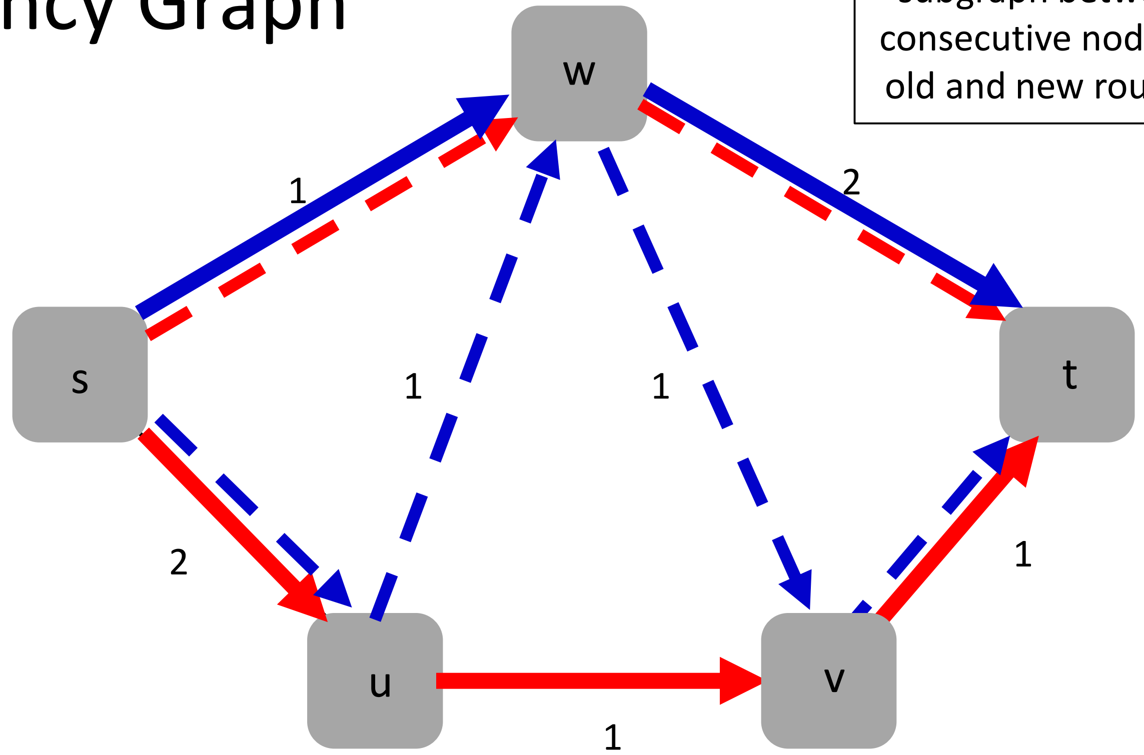
Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s
4. blue@w

Round 4

Block Decomposition and Dependency Graph

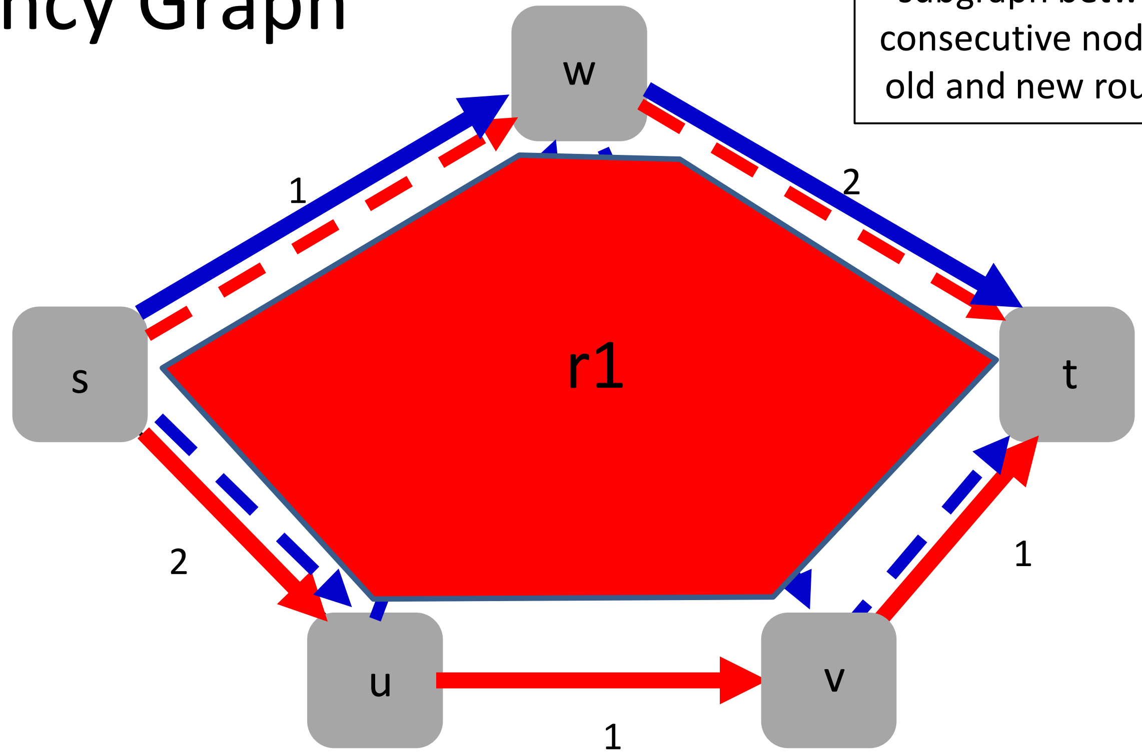
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Flow 1
Flow 2

Block Decomposition and Dependency Graph

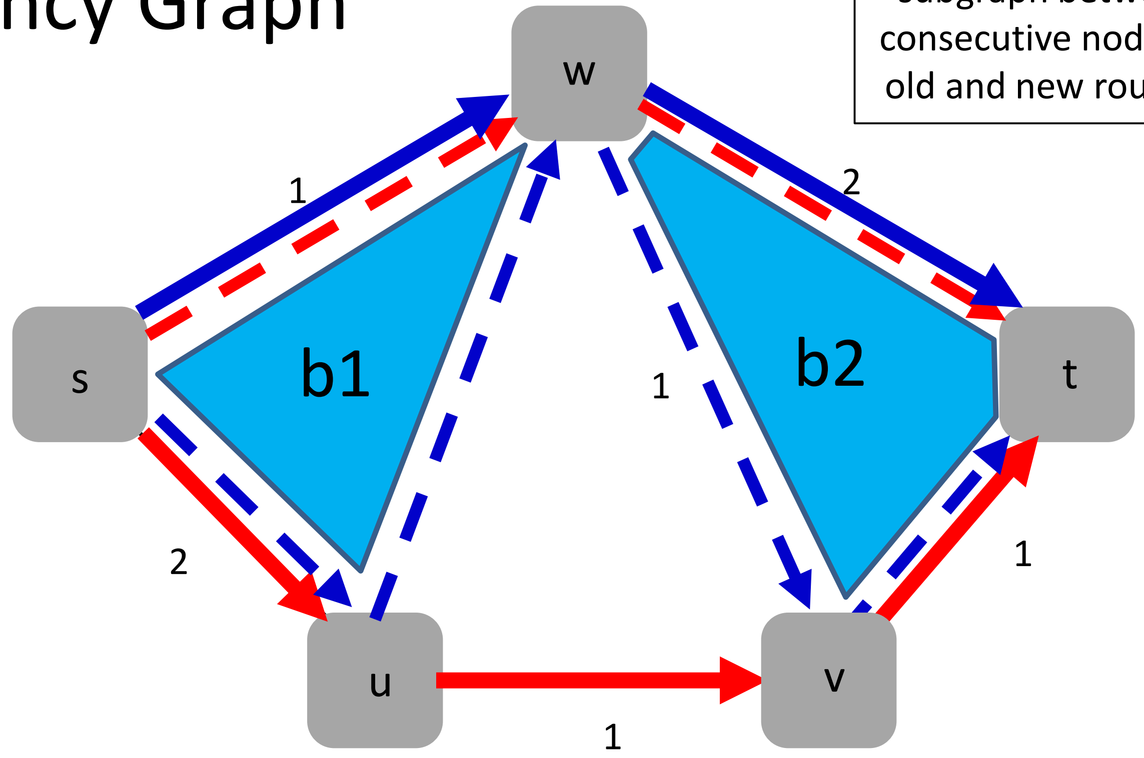
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Just one red block: r1

Block Decomposition and Dependency Graph

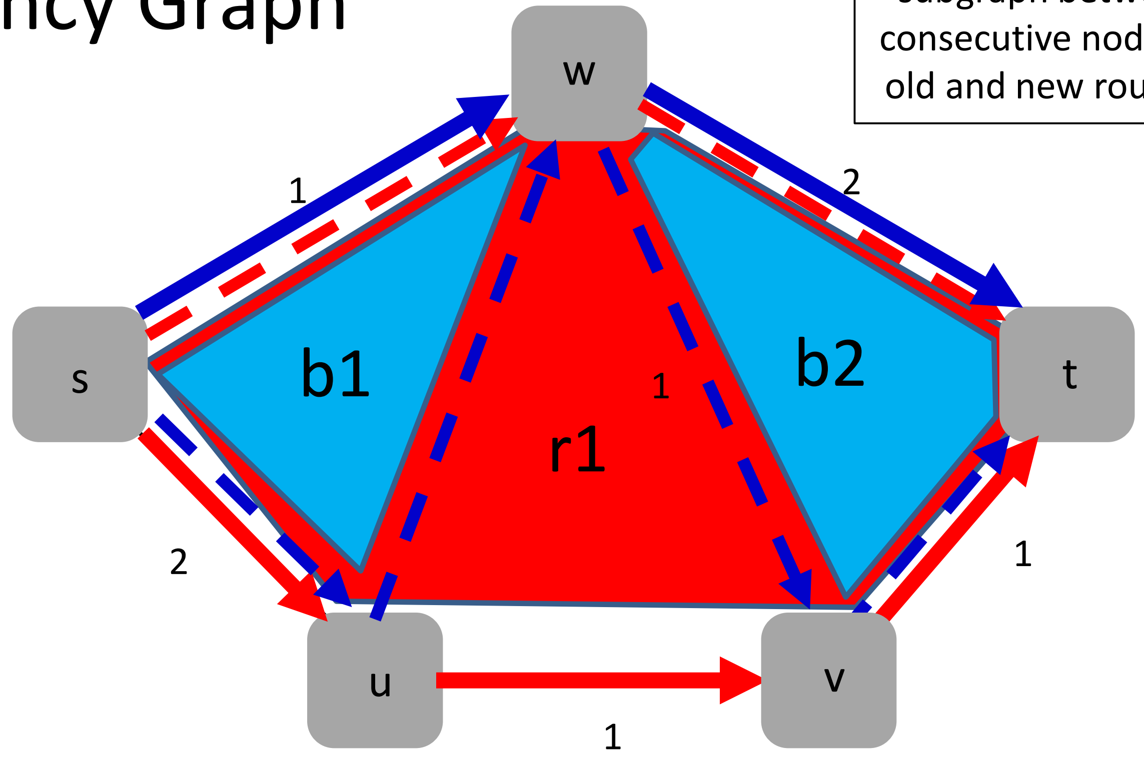
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Two blue blocks: b1 and b2

Block Decomposition and Dependency Graph

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Dependencies: update b2 after r1 after b1.

Many Open Problems

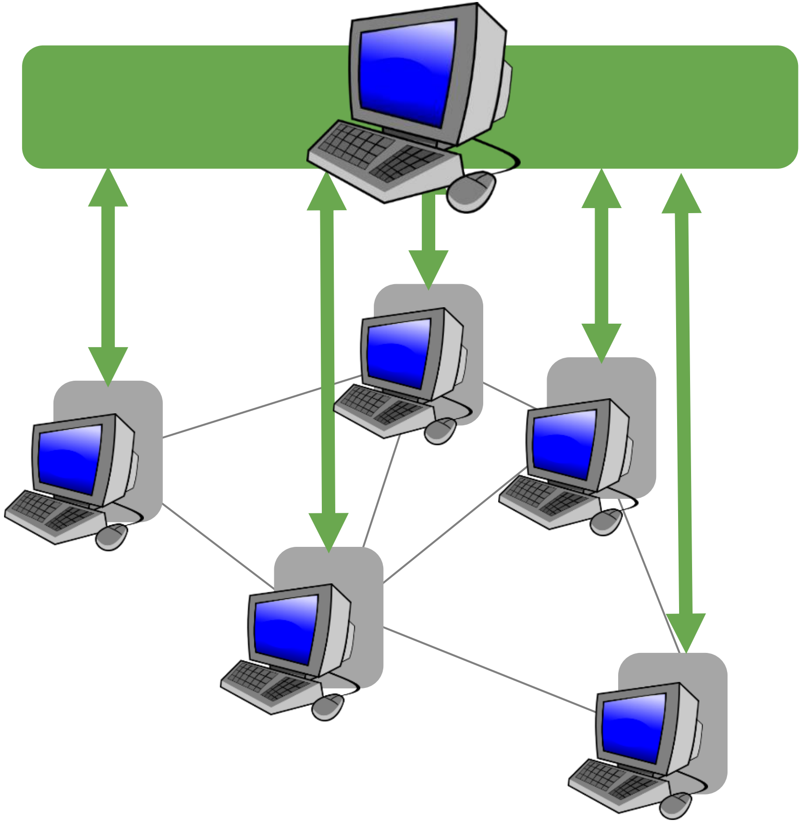
- We know for DAG:
  - For k=2 flows, polynomial-time algorithm to compute schedule with **minimal number of rounds**!
    - For general k, NP-hard
  - For general k flows, polynomial-time algorithm to compute **feasible update**

- Everything else: ***unkown!***
  - In particular: what if flow graph is not a DAG?

ACM PODC 2015
ACM SIGMETRICS 2016

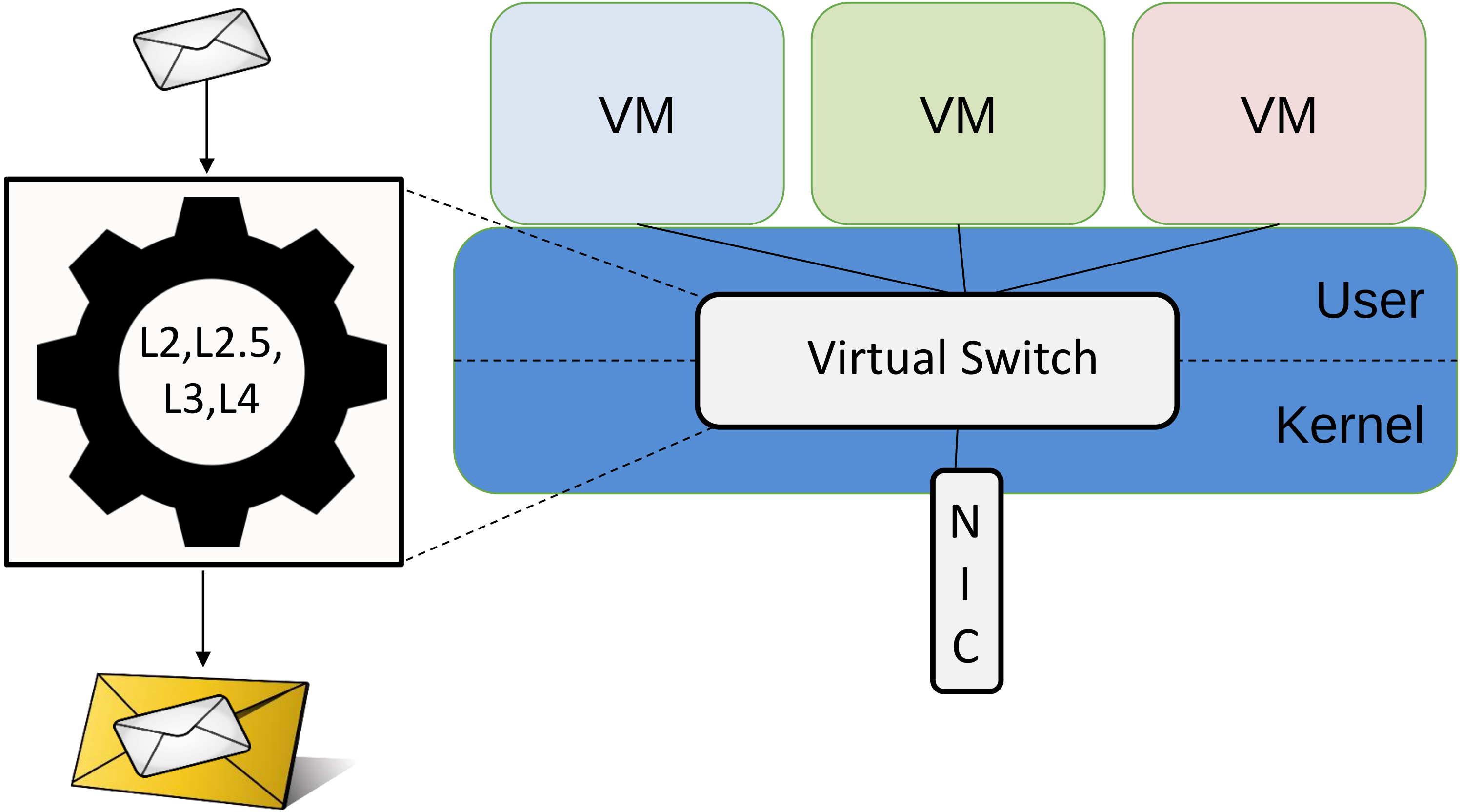
Trend: Virtualization

- Routers, switches, **middleboxes** run on commodity x86 hardware
- A.k.a. **virtual switches**
- Mainly in datacenters
- Uncharted security landscape!

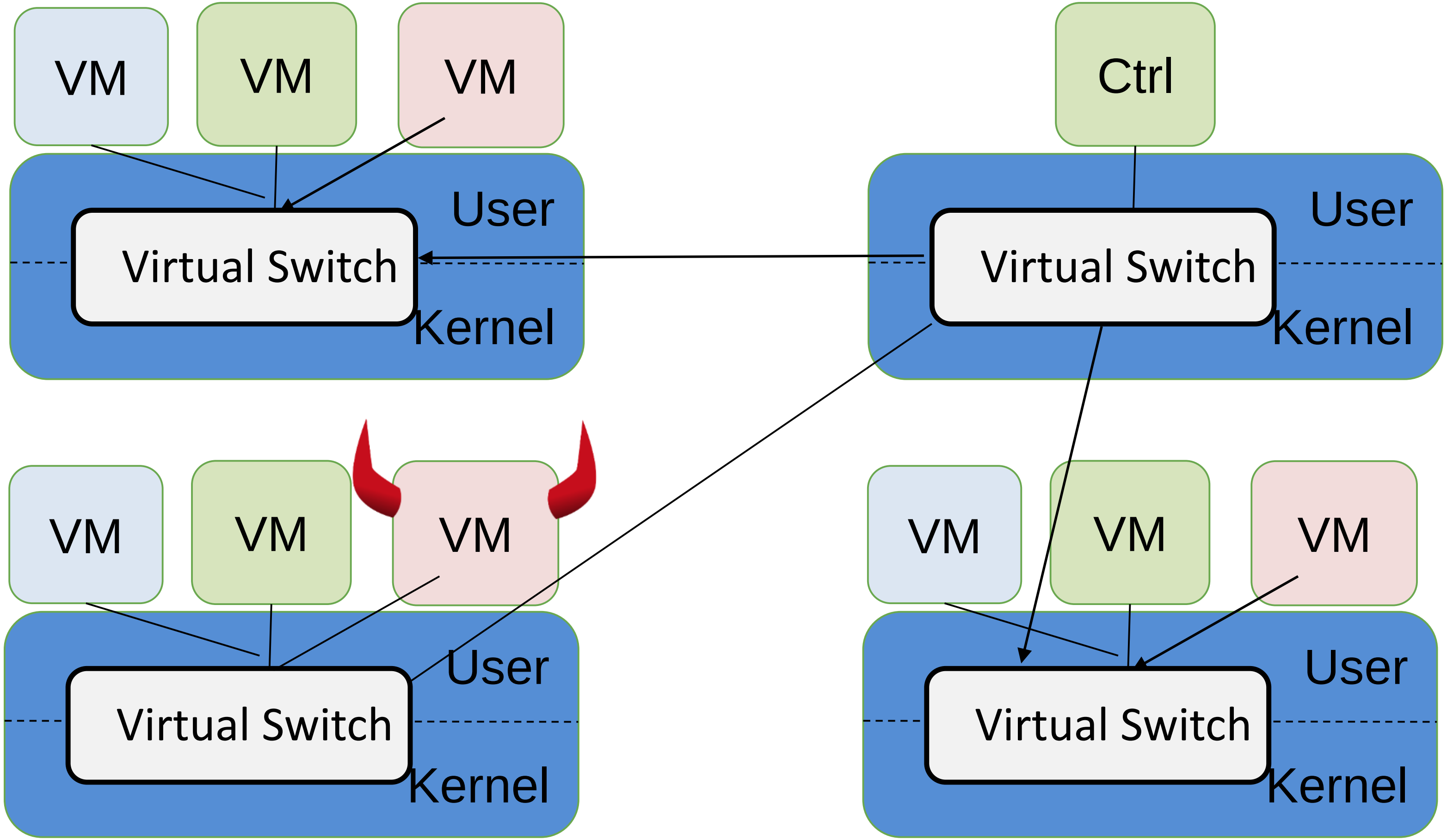


Virtual Switches are Complex, e.g.: (Unified) Packet Parsing

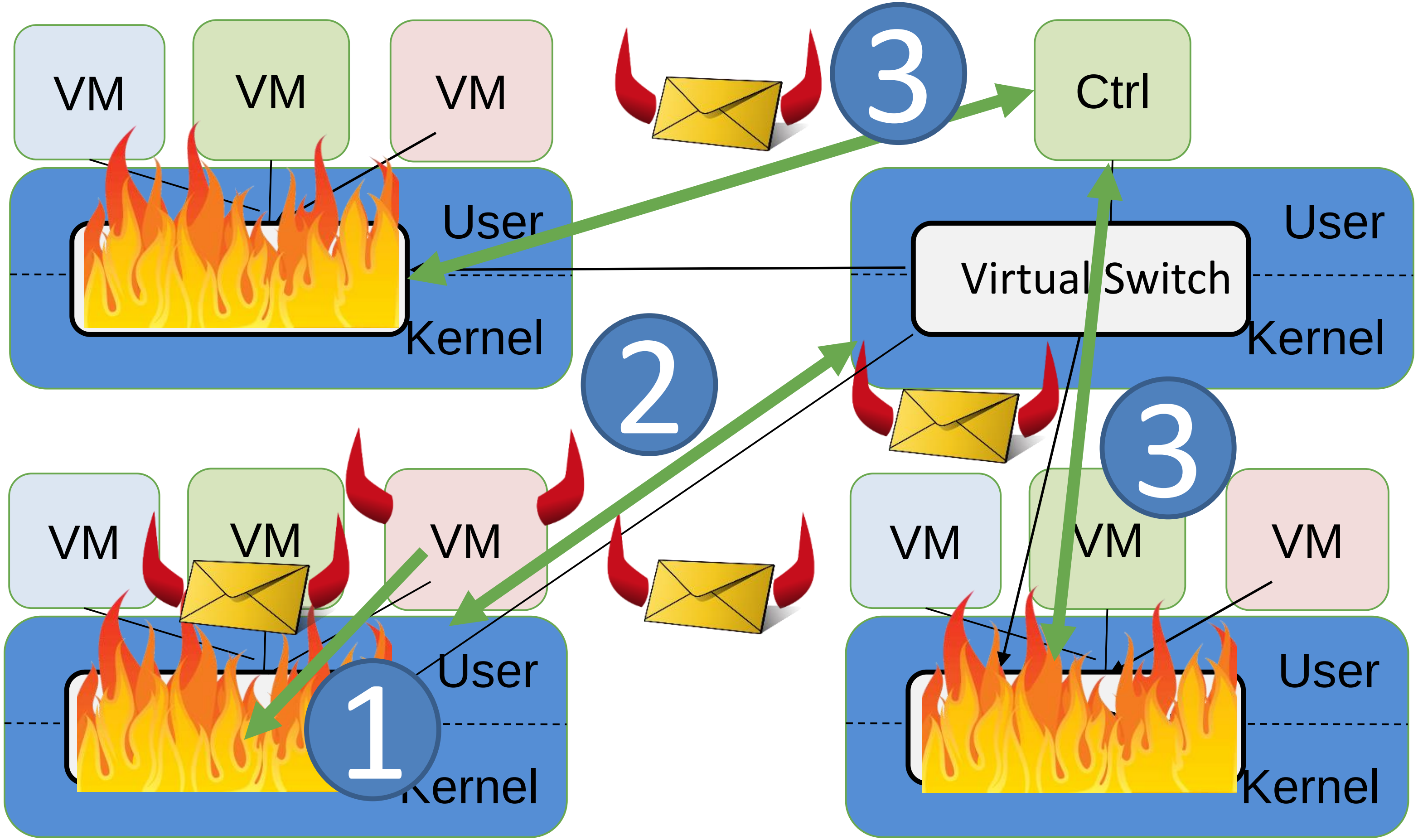
- Ethernet
- LLC
- VLAN
- MPLS
- IPv4
- ICMPv4
- TCP
- UDP
- ARP
- SCTP
- IPv6
- ICMPv6
- IPv6 ND
- GRE
- LISP
- VXLAN
- PBB
- IPv6 EXT HDR
- TUNNEL-ID
- IPv6 ND
- IPv6 EXT HDR
- IPv6HOPOPTS
- IPv6ROUTING
- IPv6Fragment
- IPv6DESTOPT
- IPv6ESP
- IPv6 AH
- RARP
- IGMP



Compromising the Cloud



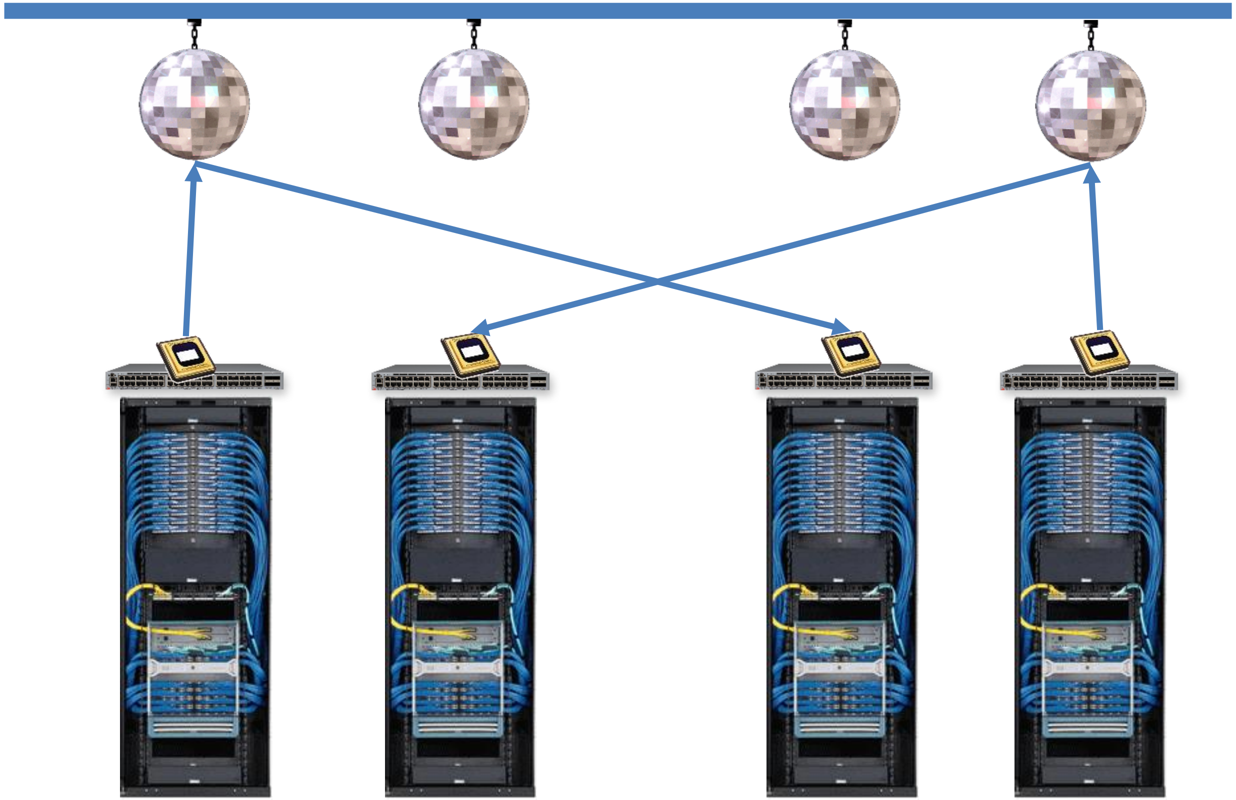
Compromising the Cloud



Today: Demand-Aware Networks (DANs) and Self-Adjusting Networks (SANs)

Today: Demand-Aware Networks (DANs) and Self-Adjusting Networks (SANs)

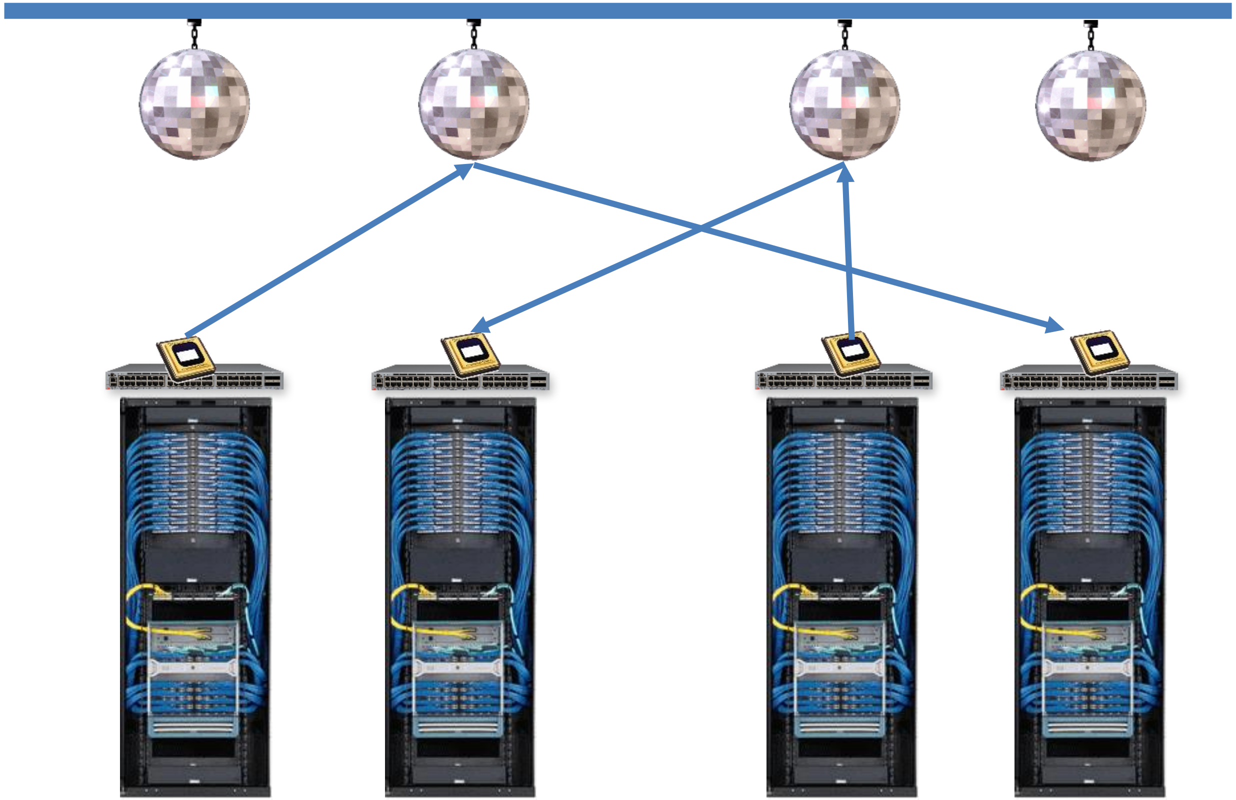
Started as a theoretical project, *but then*:



t=1

Today: Demand-Aware Networks (DANs) and Self-Adjusting Networks (SANs)

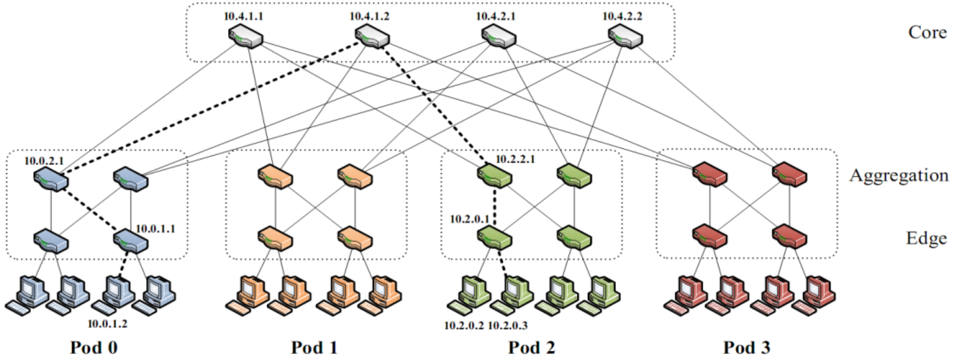
Started as a theoretical project, *but then*:



t=2

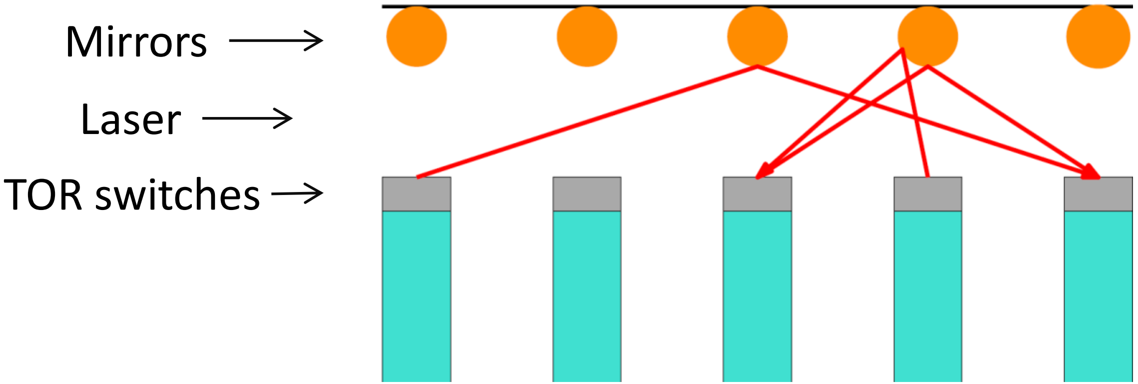
Today (still): Static Networks

- Traditional datacenter networks are **static**
  - Lower bounds and undesirable **trade-offs**, e.g., degree vs diameter
  - Usually optimized for the “worst-case” (all-to-all communication)
  - Example, fat-tree topologies: provide **full bisection bandwidth**



Next: Reconfigurable Networks?

- The physical topology becomes **reconfigurable**
  - Enables demand-aware network designs
  - Example: **ProjecToR** (SIGCOMM 2016)



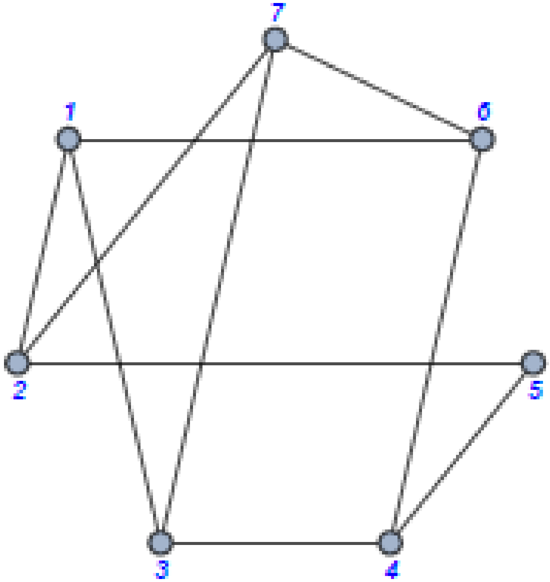
Our Research Vision:
Demand-Aware Networks (DANs)

Destinations

Sources

$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0 \\
\hline
\end{array}$$

$\Rightarrow$ design



Demand matrix: joint distribution

DAN (of constant degree)

Our Research Vision:
Deman Networks (DANs)

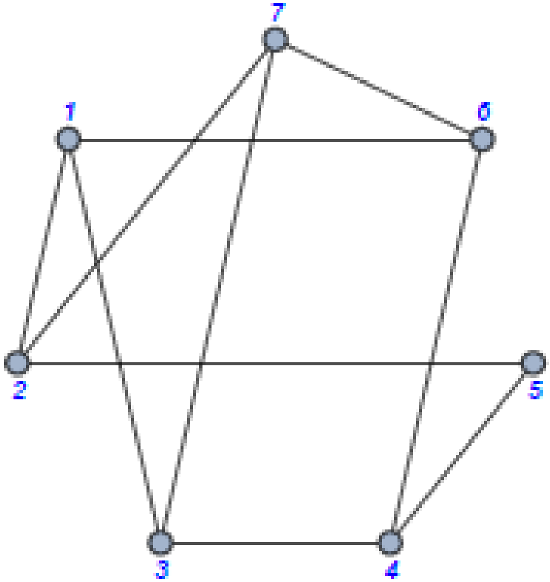
Can be seen as a graph as well: the workload!

Destinations

Sources

$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0
\end{array}$$

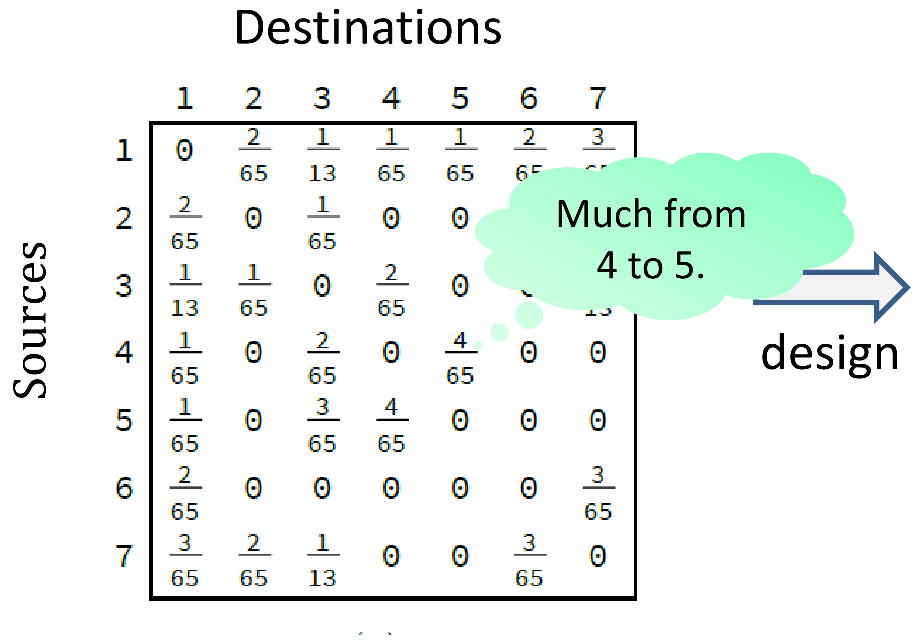
design



Demand matrix: joint distribution

DAN (of constant degree)

Our Research Vision:
Demand-Aware Networks (DANs)



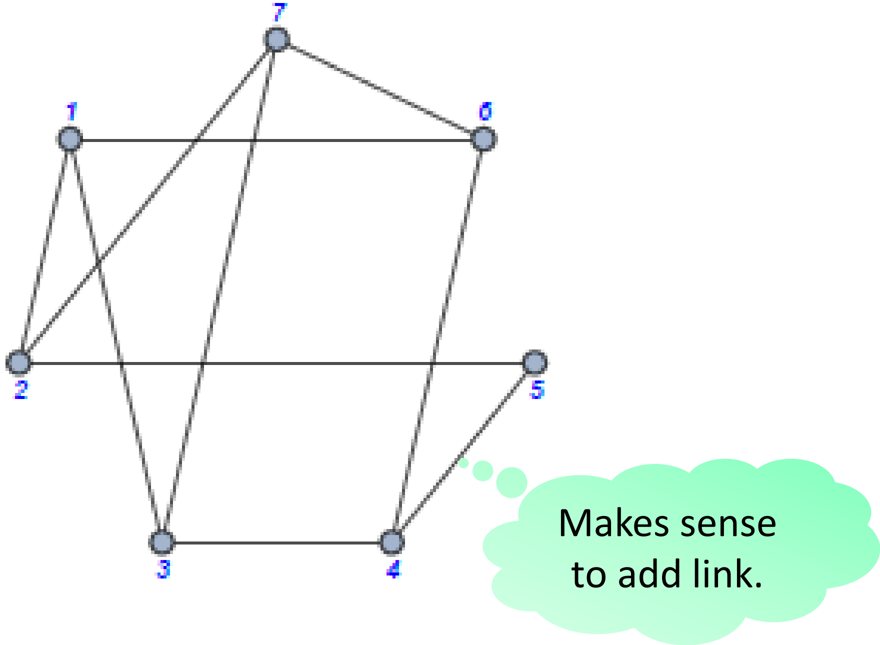
Destinations

Sources

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	[illegible]	[illegible]
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	[illegible]
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Much from
4 to 5.

design



Makes sense
to add link.

Demand matrix: joint distribution

DAN (of constant degree)

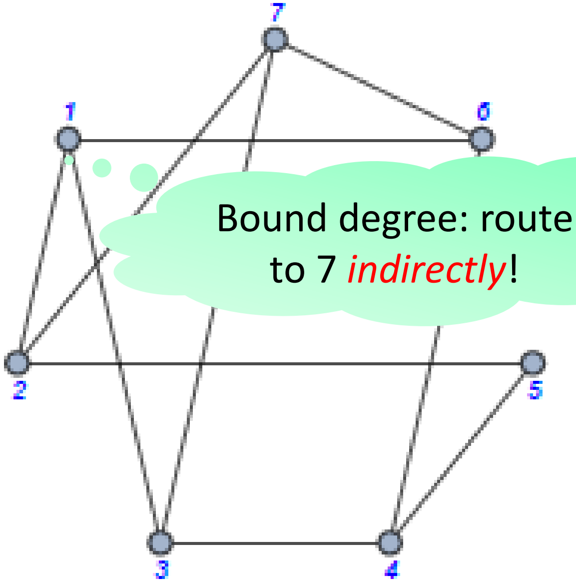
Our Research Vision:
Demand-Aware Networks (DANs)

1 communicates to many.

Sources

$$
\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0
\end{array}
$$

design



Bound degree: route to 7 *indirectly*!

Demand matrix: joint distribution

DAN (of constant degree)

Our Research Vision: Demand-Aware Networks (DANs)

Destinations

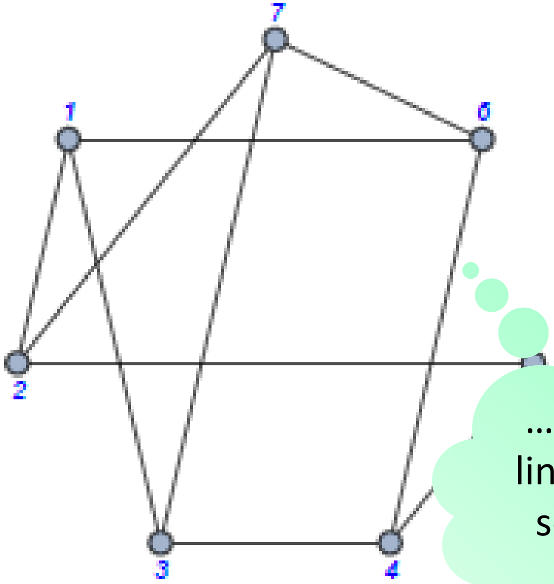
Sources

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	3	[illegible]

4 and 6 don't communicate…

Demand matrix: joi[illegible] dis[illegible]tion

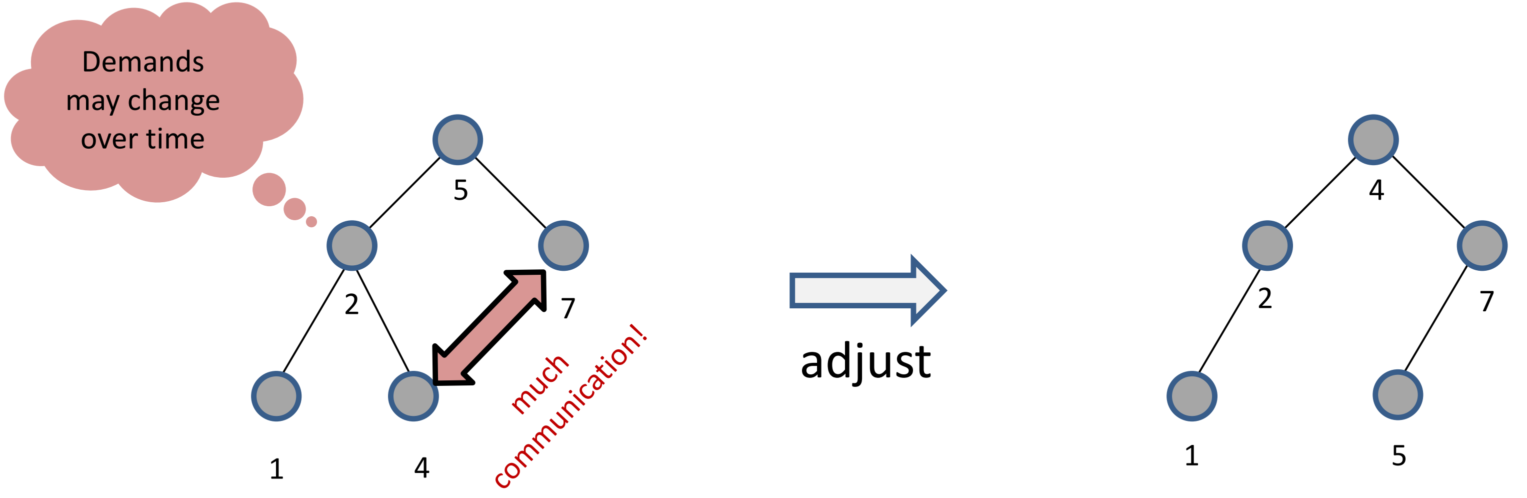
design



… but „extra" link still makes sense: not a subgraph!

DAN (of constant degree)

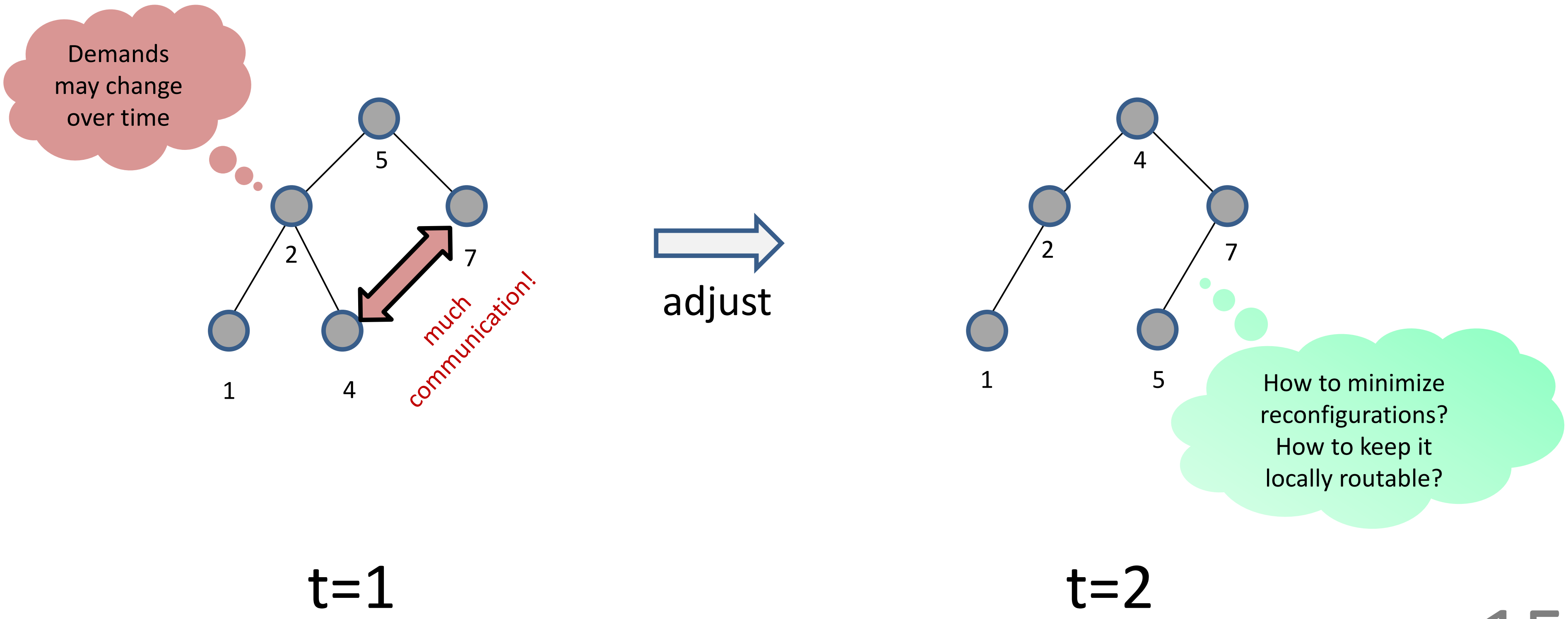
Our Research Vision:
Or Even Self-Adjusting Networks (SANs)



t=1

t=2

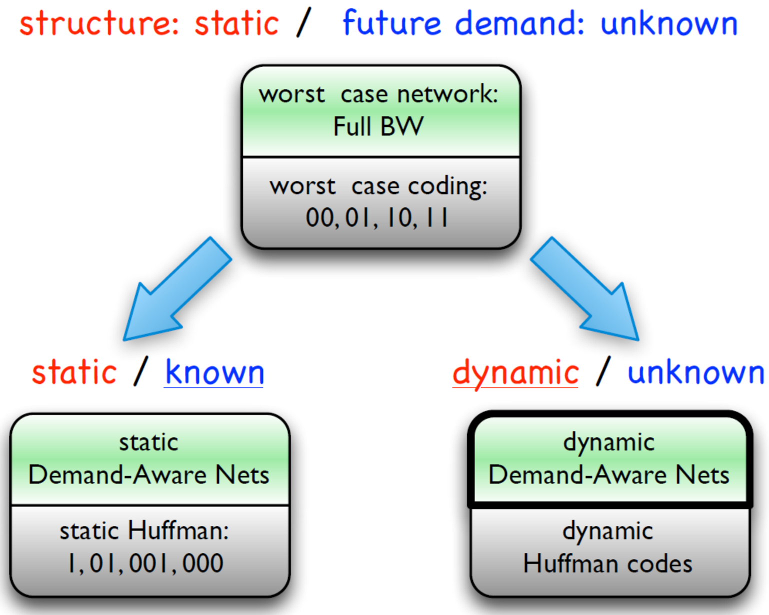
Our Research Vision:
Or Even Self-Adjusting Networks (SANs)



Demands may change over time

How to minimize reconfigurations? How to keep it locally routable?

Our Research Vision: An Analogy to Coding



structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

static / known

static
Demand-Aware Nets

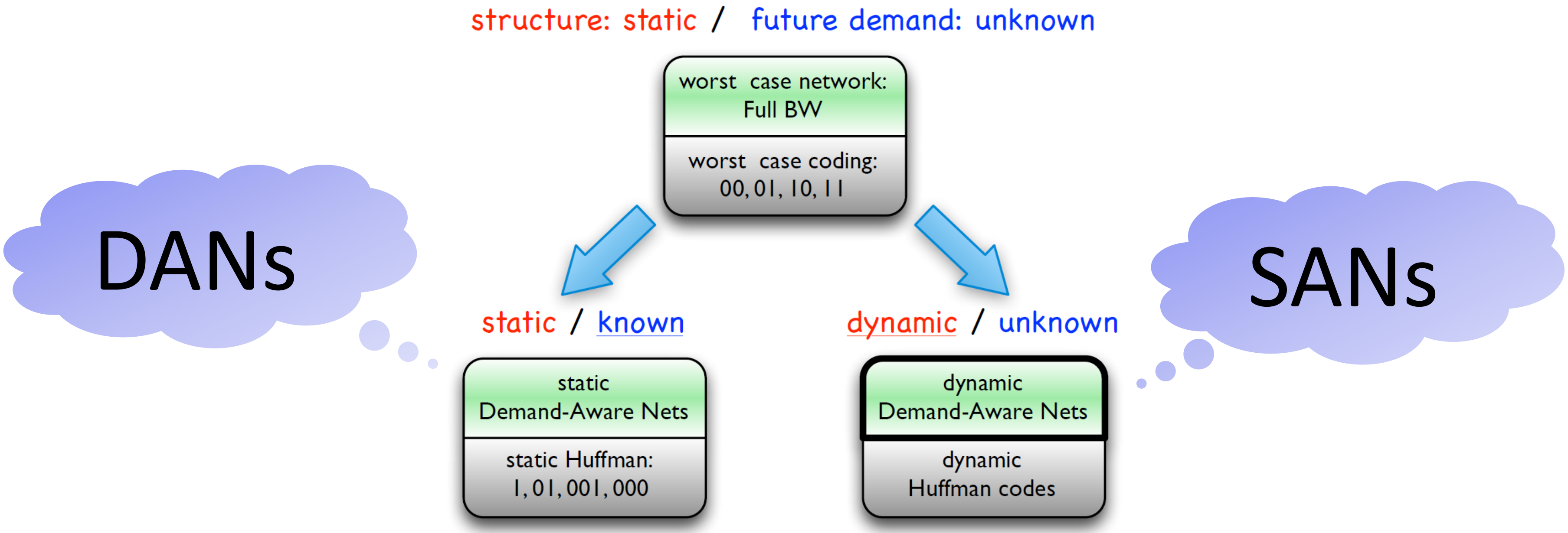
static Huffman:
1, 01, 001, 000

dynamic / unknown

dynamic
Demand-Aware Nets

dynamic
Huffman codes

Our Research Vision:
An Analogy to Coding



structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

DANs

static / known

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

SANs

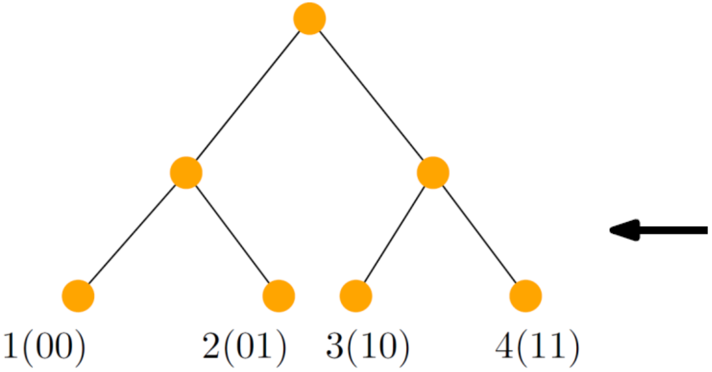
dynamic / unknown

dynamic
Demand-Aware Nets

dynamic
Huffman codes

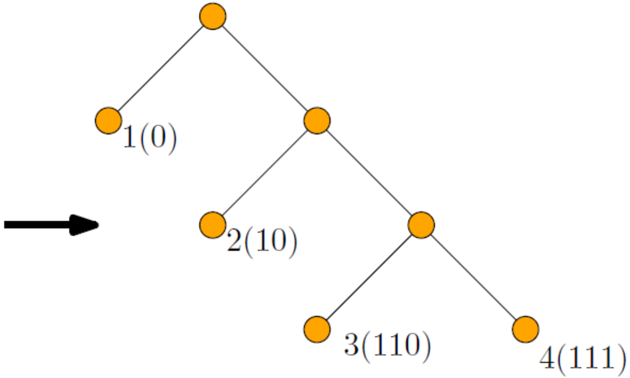
Relationship to Coding: Example

- Instead of optimizing *worst-case* performance, leverage information about specific communication pattern to design better networks.
  - Example: **Huffman Coding**.



$p(x_i)$	x_i
1/4	1
1/4	2
1/4	3
1/4	4

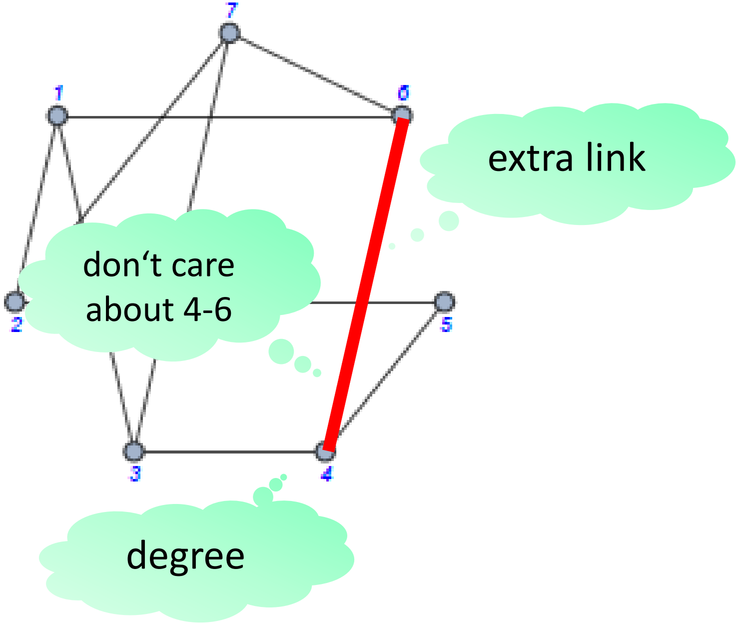
$p(x_i)$	x_i
1/2	1
1/4	2
1/8	3
1/8	4



DAN: Relationship to…

Sparse, low-distortion **graph spanners**

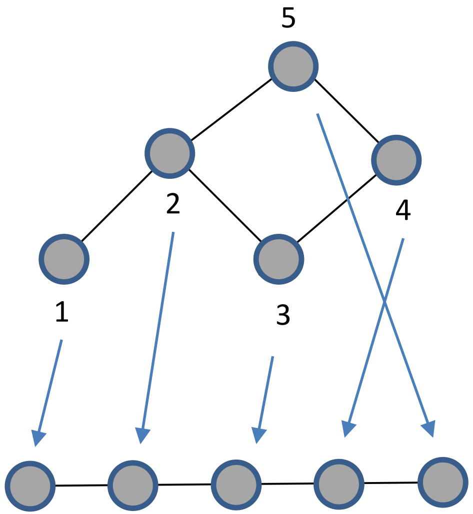
- Similar: keep distances in a „compressed network“ (few edges)
- *But:*
  - We only care about path length **between communicating nodes**, not all node pairs
  - We want **constant degree**
  - Not restricted to subgraph but can have „**additional links**“ (like geometric spanners)



DAN: Relationship to…

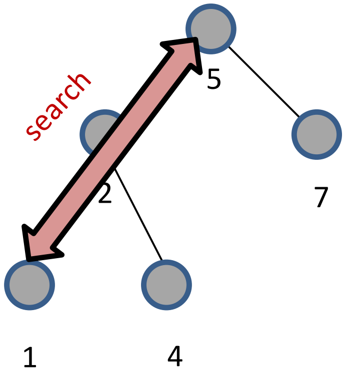
Minimum Linear **Arrangement** (MLA)

- MLA: map guest graph to line (host graph) so that sum of distances is minimal
- DAN similar: if degree bound = 2, DAN is line or ring (or sets of lines/rings)
- *But* unlike "**graph embedding problems**"
  - The host graph is also subject to optimization
  - Does this render the problem simpler or harder?

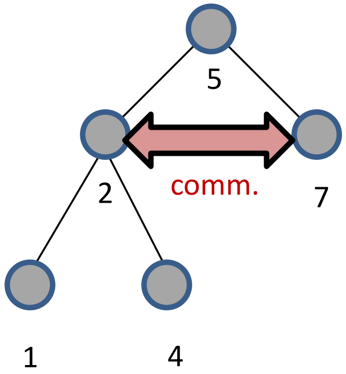


SAN: Relationship to…

- **Self-adjusting datastructures** like splay trees
- *But:* Requests are „pair-wise“, not only „from the root“



Splay Tree



SplayNet

Many interesting research questions

- How to design **static** demand-aware networks?
- How much better can demand-aware networks be compared to **demand-oblivious** networks?
- How to design **dynamic** or even **decentralized** self-adjusting demand-aware networks?

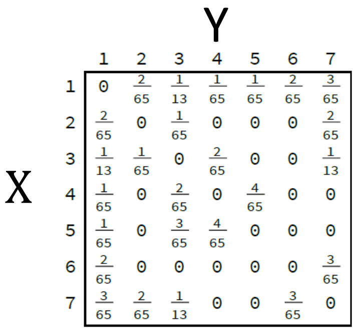
Remainder of This Talk

- Insights into Demand-Aware Networks (DANs) DISC 2017
- Insights into Self-Adjusting Networks (SANs) TON 2016
- Some words about migration… DISC 2016
- Conclusion

DANs: The Problem

Input:

$\mathcal{D}[p(i,j)]$: joint distribution, Δ



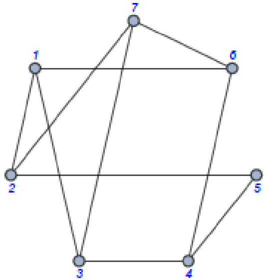
Y

X

$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0
\end{array}$$

Output:

N: DAN



Bounded degree Δ=3

Expected Path Length (EPL): Basic measure of efficiency

$$\mathrm{EPL}(\mathcal{D},N) = \mathbb{E}_{\mathcal{D}}[d_N(\cdot,\cdot)] = \sum_{(u,v)\in\mathcal{D}} p(u,v)\cdot d_N(u,v)$$

Bounded Network Design (BND)

- **Inputs**: Communication distribution $\mathcal{D}[\mathrm{p}(\mathrm{i},\mathrm{j})]_{\mathrm{nxn}}$ and a maximum degree Δ.

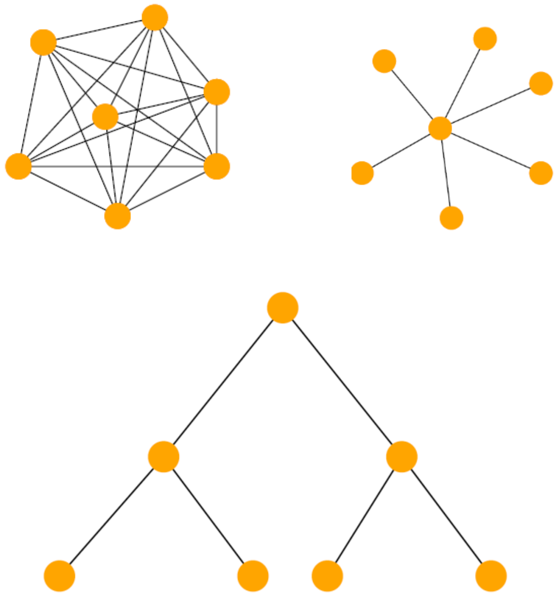
- **Output**: A Demand Aware Network $\mathrm{N} \in \mathcal{N}_{\Delta}$ s.t.

$$\mathrm{BND}(\mathcal{D}, \Delta) = \min_{\mathrm{N}\in\mathcal{N}_{\Delta}} \mathrm{EPL}(\mathcal{D},\mathrm{N})$$

Bounded degree

Some Insights

- Clique and star have constant EPL but unbounded degree.
- What about a complete binary tree?
  - Degree 3
  - $d_N(u,v) \leq 2 \log n$
  - Hence EPL $= O(\log n)$
- Can we do better than $\log n$?



An Entropy Lower Bound

- EPL related to **entropy**. Intuition:
  - Low entropy: e.g., uniform distribution, not much structure, long paths
  - High entropy: can exploit structure to create topologies with short paths

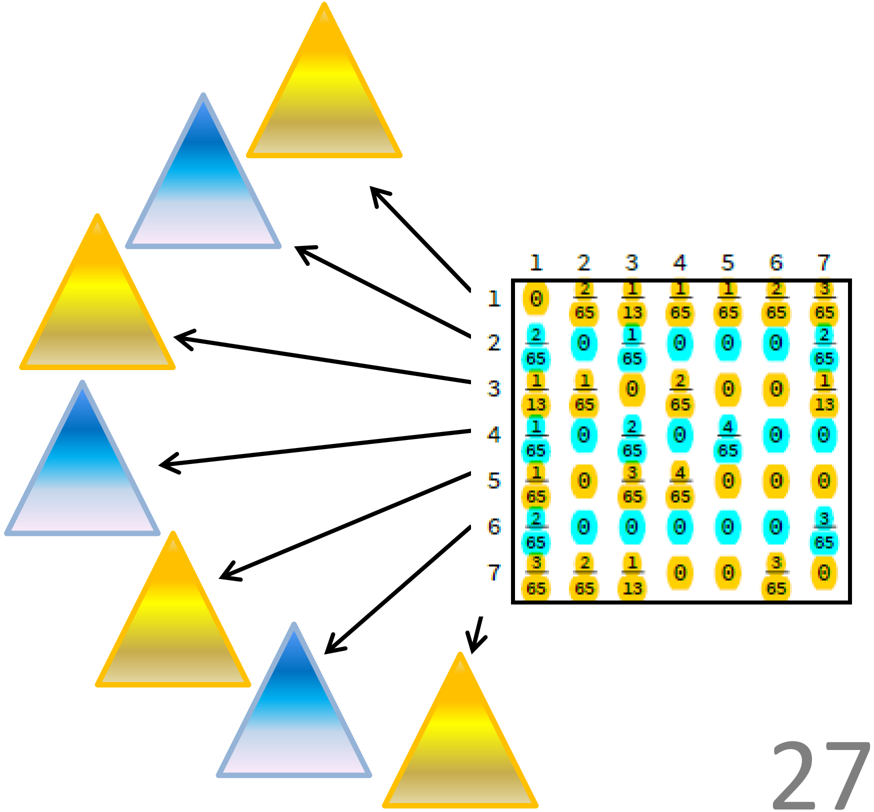
- **Theorem:** Let X, Y be the marginal distributions of the sources and destinations in $\mathcal{D}$ respectively. Then

$$\mathrm{EPL}(\mathcal{D}, \Delta) \geq \Omega(\mathrm{H}_\Delta(\mathrm{Y}|\mathrm{X}) + \mathrm{H}_\Delta(\mathrm{X}|\mathrm{Y}))$$

- **Conditional entropy**: Average uncertainty of X given Y
  - $\mathrm{H}(\mathrm{X}|\mathrm{Y}) = \sum_{i=1}^{n} p(x_i, y_j) \log_2(1/p(x_i|y_j))$

Lower Bound: Idea

- **Proof idea** (EPL=$\Omega(H_\Delta(Y|X))$):
- Build optimal Δ-ary tree for each source i: entropy lower bound known on EPL known for binary trees (Mehlhorn 1975 for BST but proof does not need search property)
- Consider union of all trees
- Violates degree restriction but valid lower bound

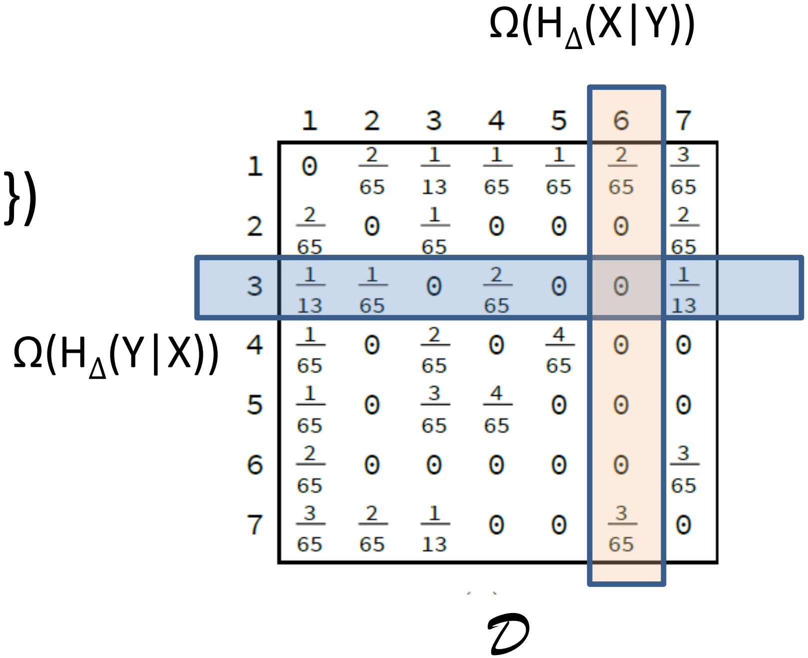


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Lower Bound: Idea

Do this in both dimensions:

$$\text{EPL} \geq \Omega(\max\{H_\Delta(Y|X), H_\Delta(X|Y)\})$$



$\Omega(H_\Delta(X|Y))$

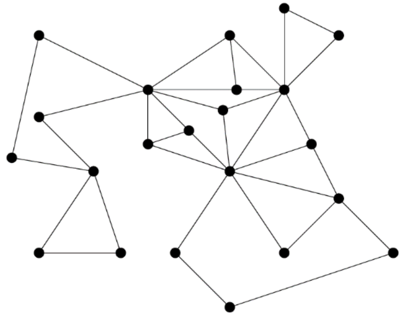
$\Omega(H_\Delta(Y|X))$

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

$\mathcal{D}$

Upper Bound: Sparse Distributions

- Real distributions are **sparse**!
  - E.g., datacentre's traffic shows that demand distributions are sparse



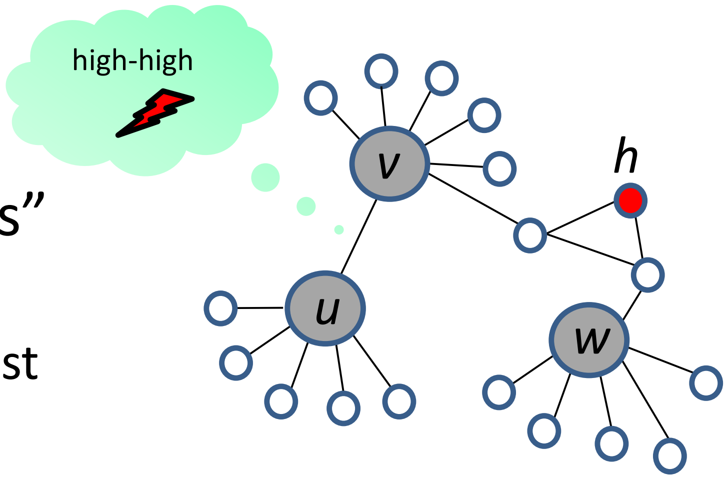
- **Theorem:** $G_{\mathcal{D}}$ is a sparse graph with constant average degree Δ_{avg} , then it is possible to find a DAN N with maximum degree 12Δ_{avg} , such that

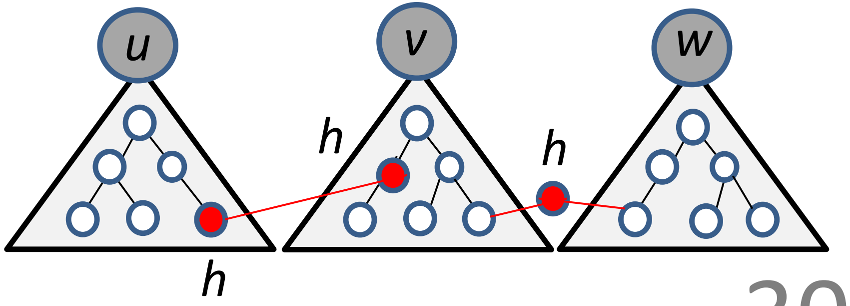
$$\mathrm{EPL}(\mathcal{D},N) \leq O(H(Y|X)+H(X|Y))$$

Asymptotically optimal

Sparse Distributions: Construction

- Idea similar to lower bound: “**union** of bounded-degree trees”
  - However, reduce degree: leverage fact that sparse graphs have at least n/2 constant-degree nodes (low-degree nodes)
  - Use them as **helper** nodes between two “large” (i.e., high-degree) nodes
  - Sparse: there are enough helper nodes,





Many Open Questions

- Demand-aware bounded doubling dimension graphs?

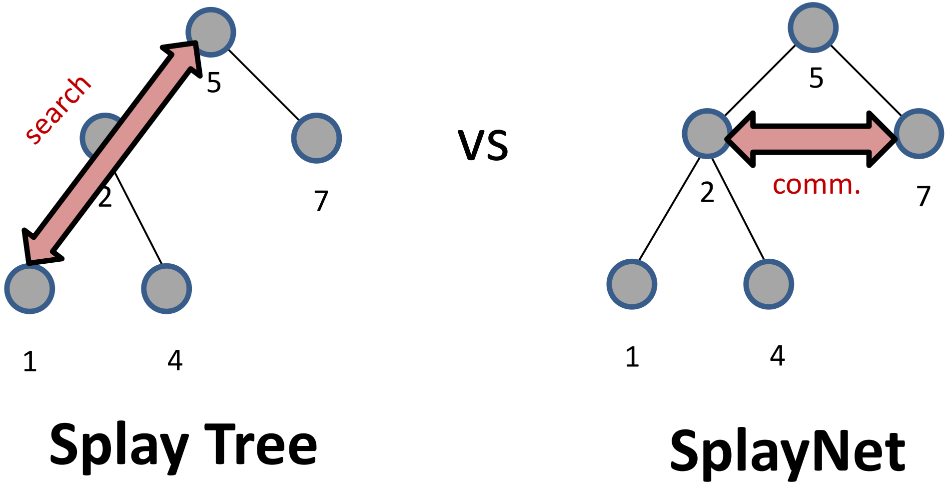
- Demand-aware continuous-discrete graphs?
  - Shannon-Fano-Elias coding

- Demand-aware skip graphs?

- …

SANs: Example "SplayNet"

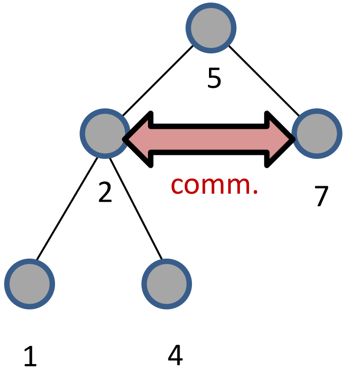
- Recall:



- Also related: Move-to-Front

Desirable Properties

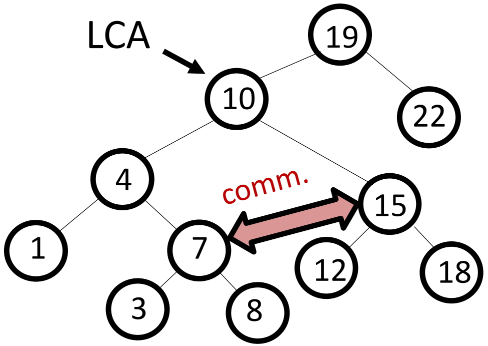
- Bounded degree
- Supports **local routing**
- "Good over time": account for **reconfiguration** costs
- **Decentralized**
- …



SplayNet

SAN Idea 1: SplayNet

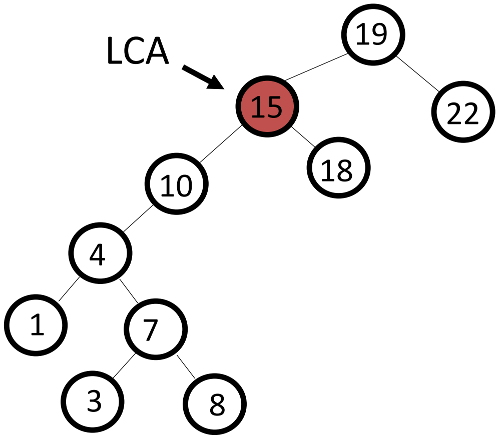
- Idea: Binary Search Tree (**BST**) *network*
- Supports local routing
  - Left child, right child, upward?
- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag
- But splay only to **Least Common Ancestor (LCA)**



SplayNet

SAN Idea 1: SplayNet

- Idea: Binary Search Tree (**BST**) *network*
- Supports local routing
  - Left child, right child, upward?
- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag
- But splay only to **Least Common Ancestor (LCA)**



SplayNet

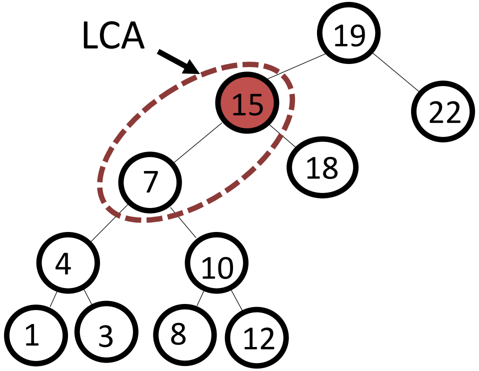
SAN Idea 1: SplayNet

- Idea: Binary Search Tree (**BST**) *network*

- Supports local routing
  - Left child, right child, upward?

- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag

- But splay only to **Least Common Ancestor (LCA)**

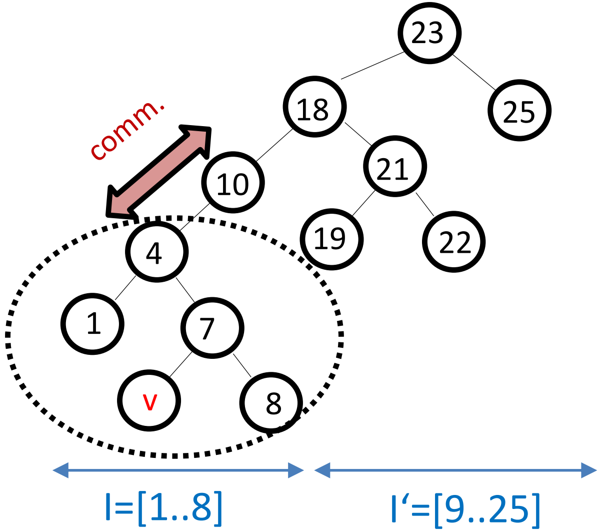


SplayNet

SplayNet: Properties

Property 1: Optimal static network can be computed in polynomial-time (dynamic programming)

- **Unlike unordered tree?**



1. Define: flow out of interval I

$W_I(v)=\sum_{u \in I'} w(u,v) + w(v,u)$

Decouple cost to ouside: distance to root of T_I only

2. Cost of a given tree T_I on I:

$\text{Cost}(T_I, W_I)=[\sum_{u,v \in I}(d(u,v)+1)w(u,v)\ ] + D_I * W_I$

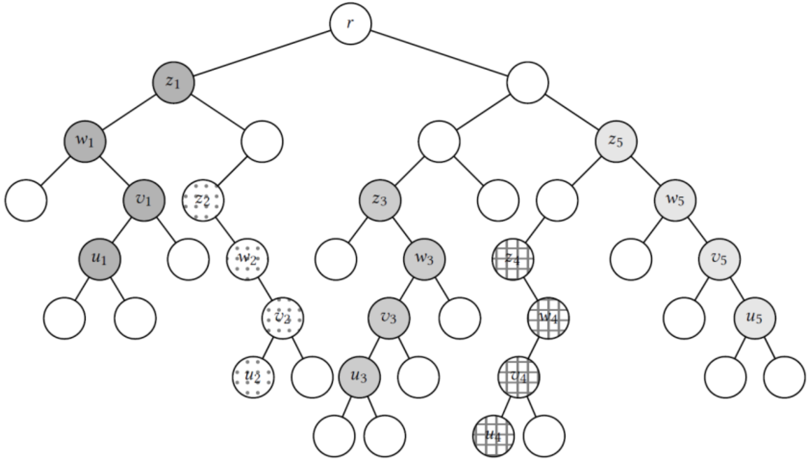
(D_I distances of nodes in I from root of T_I)

3. Dynamic program over intervals

Choose optimal root and add dist to root

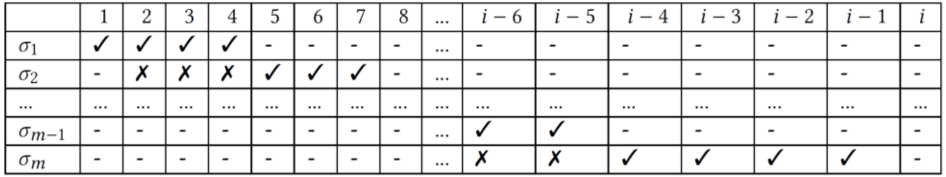
SplayNet: Properties

Property 2: Provides amortized cost and amortized throughput guarantees



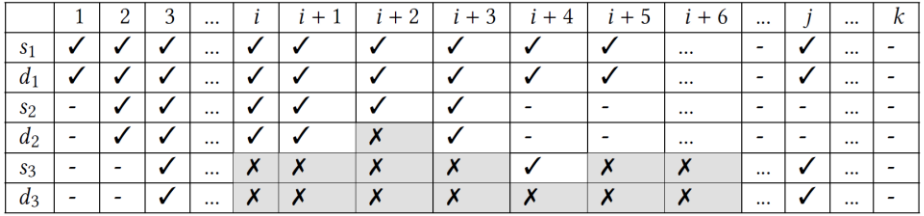
Rotations can happen concurrently: independent clusters

Splay tree: requests one after another



	1	2	3	4	5	6	7	8	...	$i-6$	$i-5$	$i-4$	$i-3$	$i-2$	$i-1$	i
σ_1	✓	✓	✓	✓	-	-	-	-	...	-	-	-	-	-	-	-
σ_2	-	✗	✗	✗	✓	✓	✓	-	...	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
σ_{m-1}	-	-	-	-	-	-	-	-	...	✓	✓	-	-	-	-	-
σ_m	-	-	-	-	-	-	-	-	...	✗	✗	✓	✓	✓	✓	-

SplayNet: concurrent

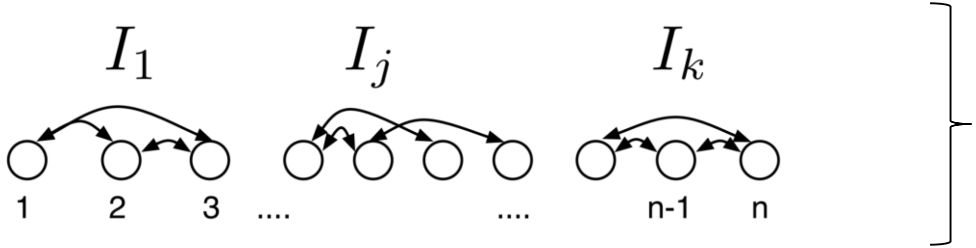


	1	2	3	...	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$	...	j	...	k
s_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
d_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
s_2	-	✓	✓	...	✓	✓	✓	✓	-	-	...	-	-	...	-
d_2	-	✓	✓	...	✓	✓	✗	✓	-	-	...	-	-	...	-
s_3	-	-	✓	...	✗	✗	✗	✗	✓	✗	✗	...	✓	...	-
d_3	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

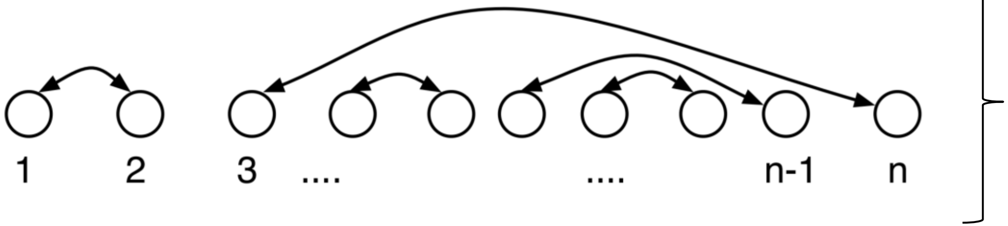
Analysis more challenging: potential function sum no longer telescopic. One request can "push-down" another.

SplayNet: Properties

Property 3: Converges to optimal network under specific demands



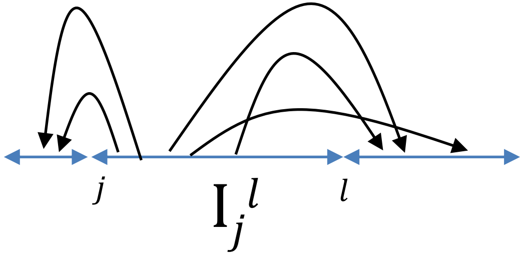
Cluster scenario: SplayNet will converge to state where path between cluster nodes only includes cluster nodes



Non-crossing matching scenario: SplayNet will converge to state where all communication pairs are adjacent

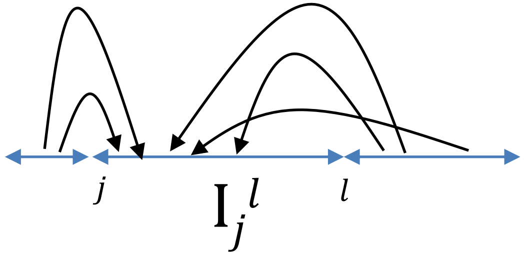
SplayNet: Improved Lower Bounds

Interval Cuts Bound



$\text{cut}_{\text{out}}(I_j^l)$

$\text{cut}_{\text{in}}(I_j^l)$



$\text{Cost} = \Omega(\max_i \min_{j,l} H(\text{cut}_{\text{in}}(I_j^l)))$

$\text{Cost} = \Omega(\max_i \min_{j,l} H(\text{cut}_{\text{out}}(I_j^l)))$

Edge Expansion Bound

- Let cut W(S) be weight of edges in cut (S,S’) for a given S
- Define a distribution w_S (u) according to the weights to all possible nodes v:

$$w_S(u) = \sum_{\substack{(u,v)\in E(S,\bar{S})\\ u\in S}} w(u,v)/W(S)$$

- Define entropy of cut and src(S),dst(S) distributions accordingly: :

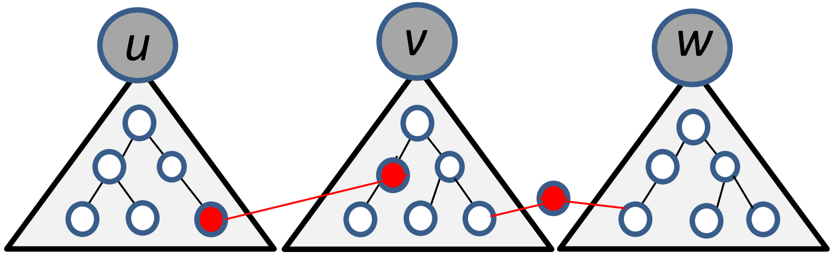
$$\varphi_H(S) = W(S)\,(H(\text{src}(S)) + H(\text{dst}(S)))$$

- Conductance entropy is lower bound:

$$\Omega(\phi_H(\mathcal{R}(\sigma)))$$

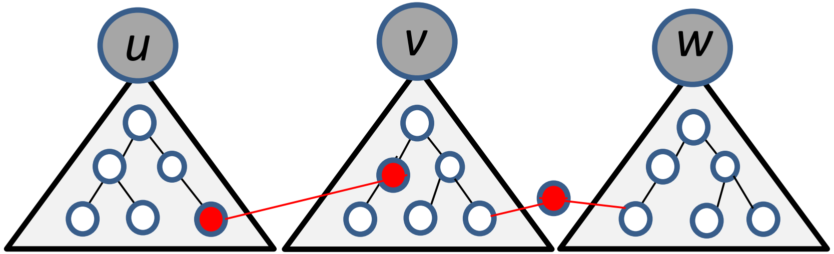
SAN Idea 2: “Splay Tree DAN”

- Recall:
  - DAN: Union of per-node binary (search) trees



Binary tree, BST, Huffman tree etc.

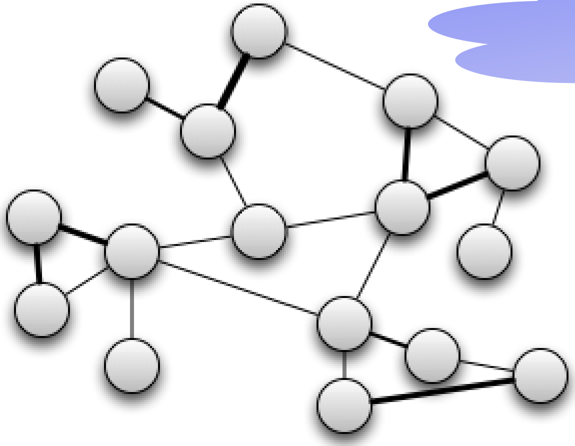
- Idea for SAN:
  - Replace per-node BST with per-node splay tree



Splay trees!

Another Dimension of Flexibility: Migration

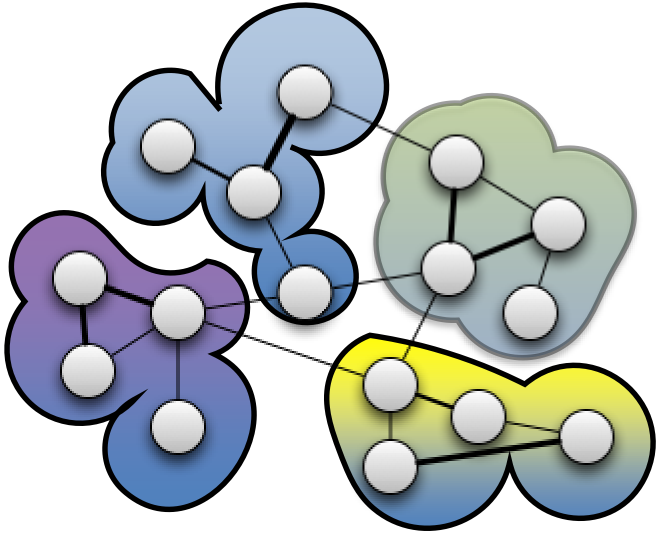
- Reduce communication cost by online *re*partitioning



How to embed communication pattern across
l=4 servers (or racks, pods, etc.) of size k=4?

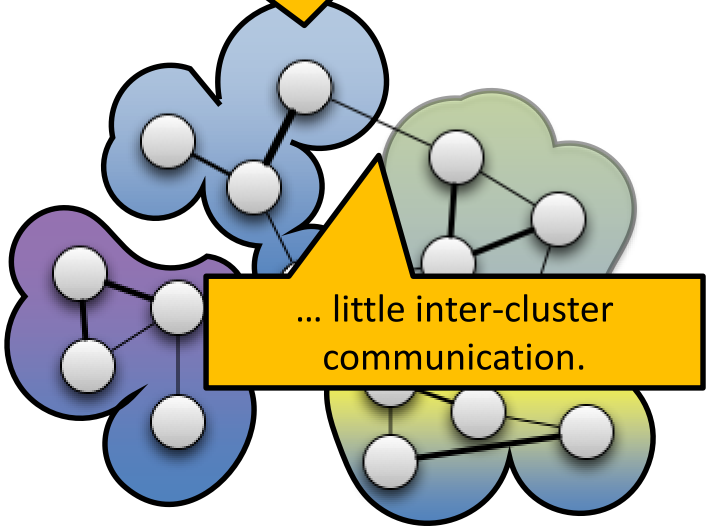
Another Dimension of Flexibility: Migration

- Reduce communication cost by online *re*partitioning



Another Dimension of Flexibility: Migration

nication cost by online *re*partitioning



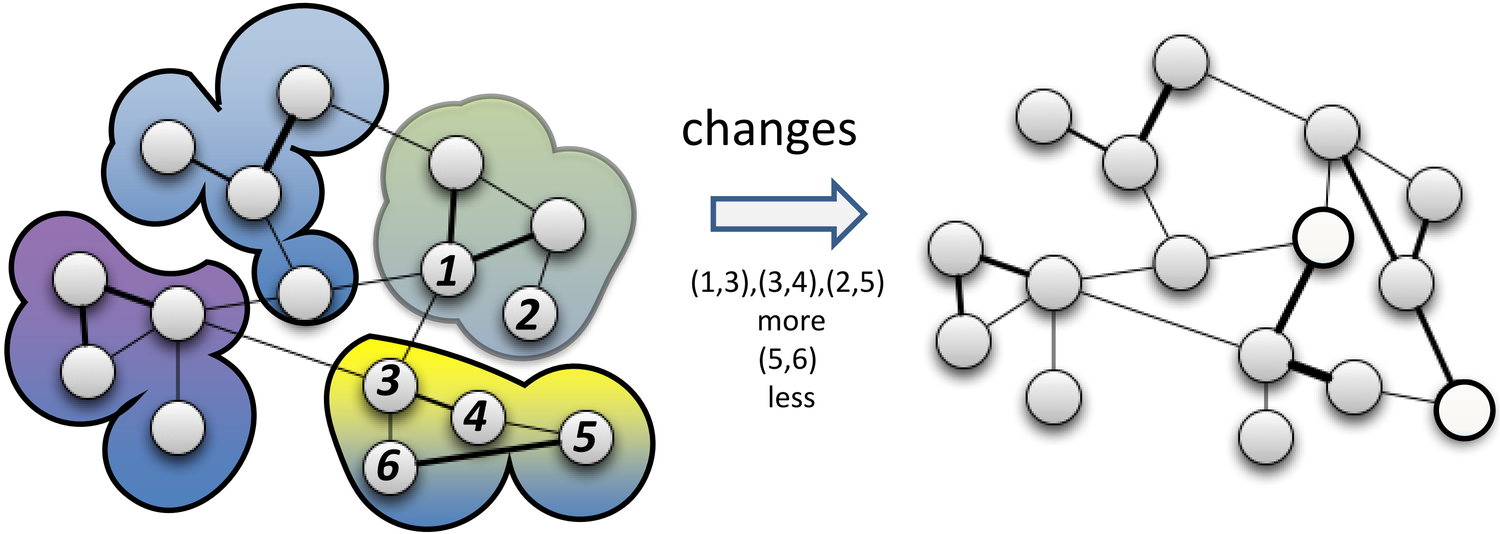
Most communication within cluster (intra-cluster)…

… little inter-cluster communication.

A classic (hard) combinatorial problem!

Another Dimension of Flexibility: Migration

- Reduce communication cost by online *re*partitioning



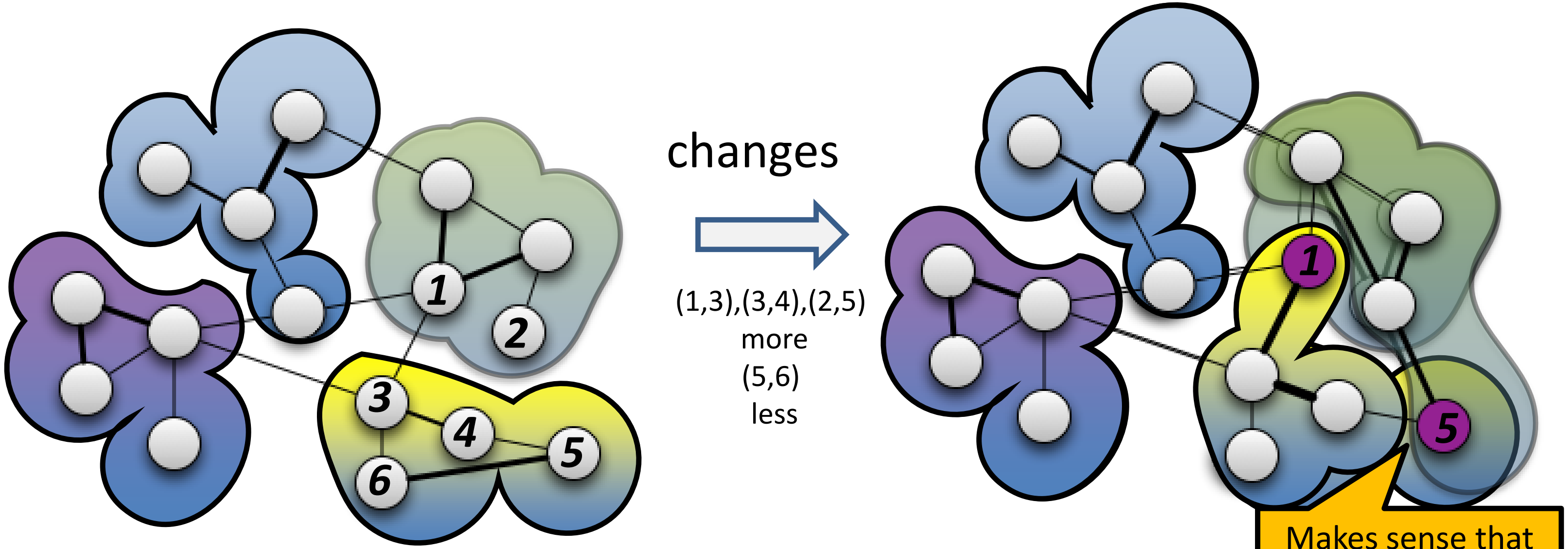
changes

(1,3),(3,4),(2,5)
more
(5,6)
less

Communication patterns can change!

Another Dimension of Flexibility: Migration

- Reduce communication cost by online *re*partitioning



changes

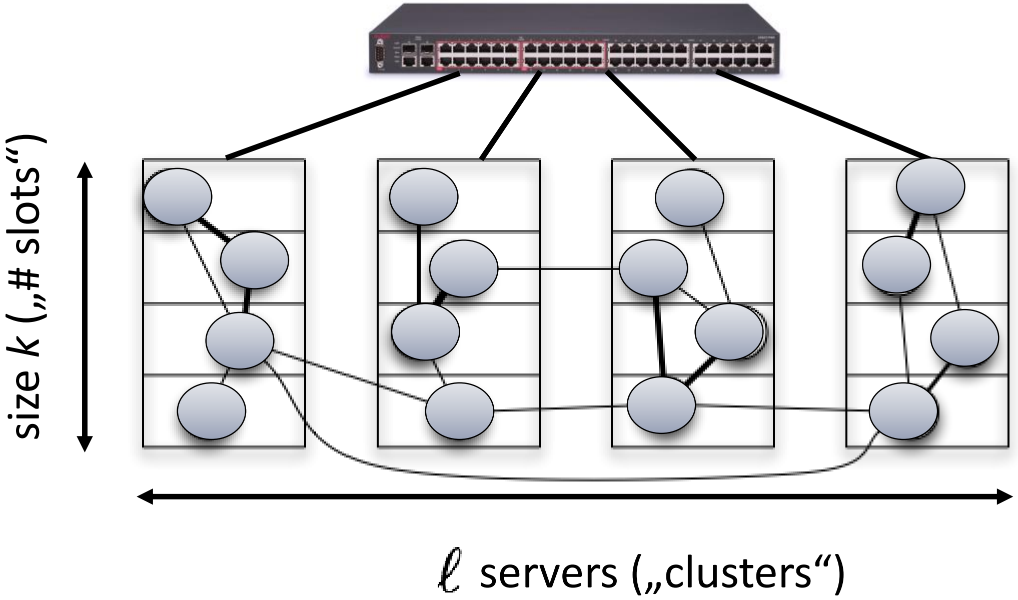
(1,3),(3,4),(2,5)
more
(5,6)
less

Communication patterns can change!

Makes sense that nodes 1 and 5 change clusters!

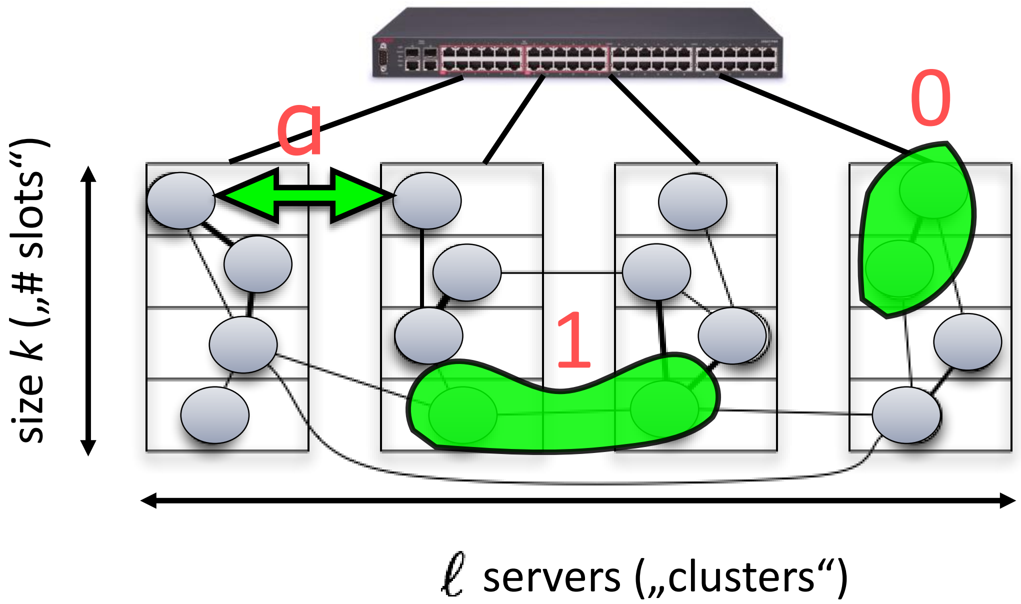
A Simple Model

- A single switch network:



A Simple Model

- A single switch network:



Adversary Models

Weak adversary

- Chooses **request distribution D**
- Requests **sampled i.i.d.** from D
- Cannot react to online algo

Strong adversary

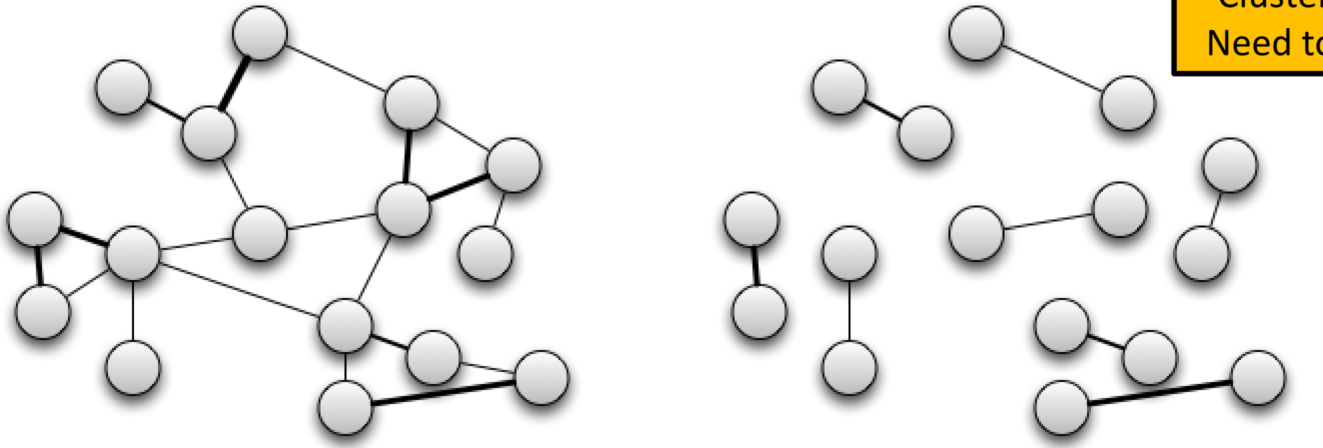
- Can generate **arbitrary request sequence σ**
- Knows and can react to online algo

The Crux: Do not know D resp. σ ahead of time

Upon each communication request (*u*,*v*):

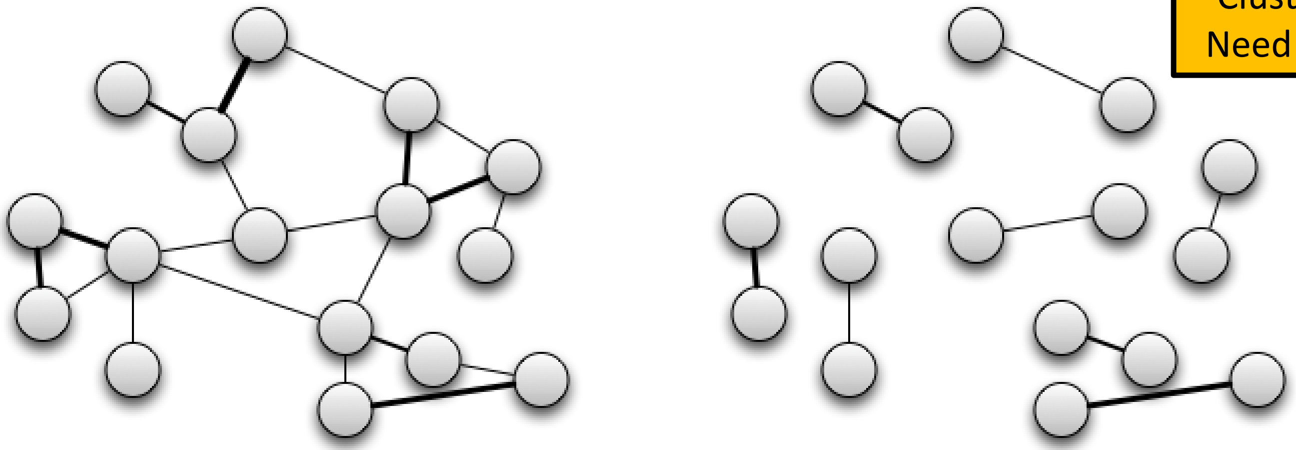
- Migrate *u* and *v* together? «Rent-or-buy»:
- Migrate where? *u* to *v*, *v* to *u*, both to a third cluster?
- If cluster is full already: what to evict?

Example: Special Case k=2



Clusters of size 2:
Need to find pairs!

Example: Special Case k=2



Clusters of size 2:
Need to find pairs!

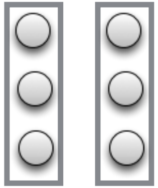
Clusters of size 2: A new type of online *re*matching problem!

It is hard to compete under  !

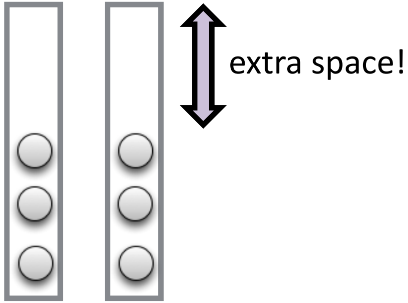
- Assume two clusters: for offline algorithm they are of size k…

- … whereas online algorithm can use clusters of size 2k-1 even (augmentation)!

OFF:

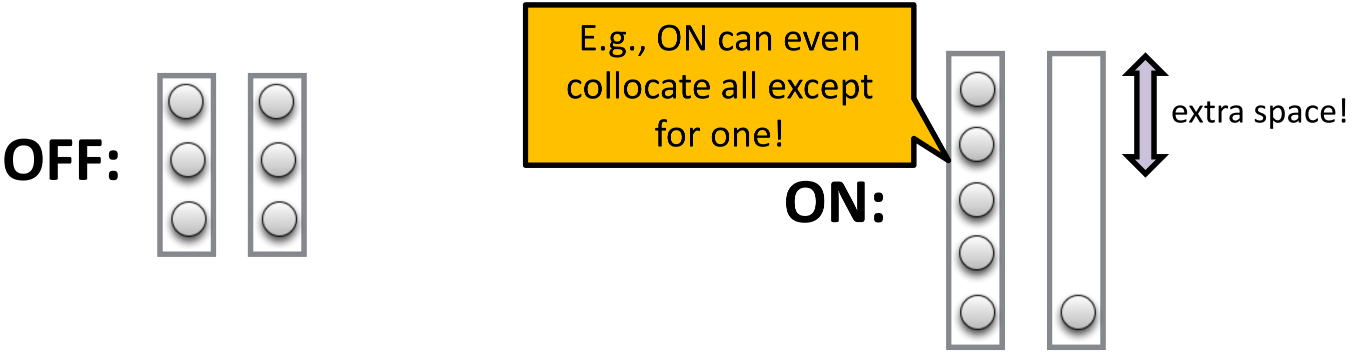


ON:



It is hard to compete under  !

- Assume two clusters: for offline algorithm they are of size k…
- … whereas online algorithm can use clusters of size 2k-1 even (augmentation)!



OFF:

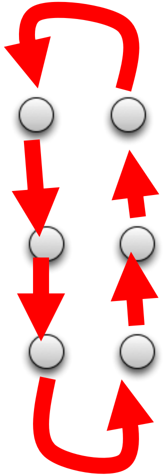
E.g., ON can even collocate all except for one!

ON:

It is hard to compete under  !

- Assume two clusters: for offline algorithm they are of size k…

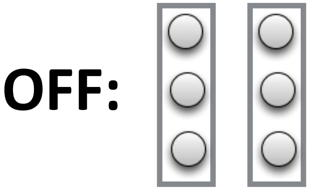
- … whereas online algorithm can use clusters of size 2k-1 even (augmentation)!



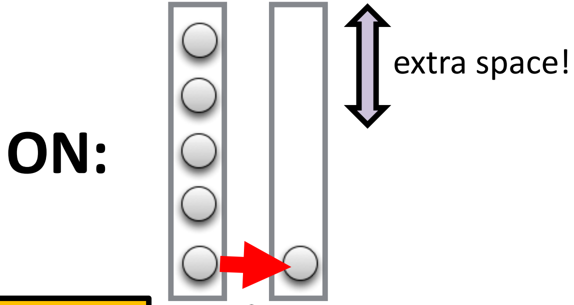
For the sake of lower bound, let us restrict the adversary more: can only ask for node pairs taken from a cyclic order: *k* pairs (resp. links) in total!

It is hard to compete under  !

- Assume two clusters: for offline algorithm they are of size k…
- … whereas online algorithm can use clusters of size 2k-1 even (augmentation)!



What is the cost of OFF?



Ouch! Cost 1 for each request.

Adversary can always request an inter-cluster link: always exists!

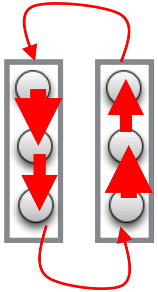
Note: adversarial strategy only depends on ON. So ON cannot learn anything about OFF!

It is hard to compete under !

- Assume two clusters: for offline algorithm they are of size k…

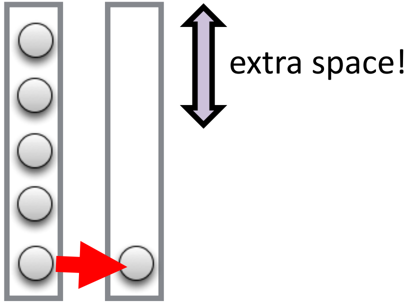
- … whereas online algorithm can use clusters of size 2k-1 even (augmentation)!

OFF:



Move to configuration i є {1,…,k} which is asked the least. Averaging argument: At least k times less communiation cost!

ON:



Adversary can always request an inter-cluster link: always exists!

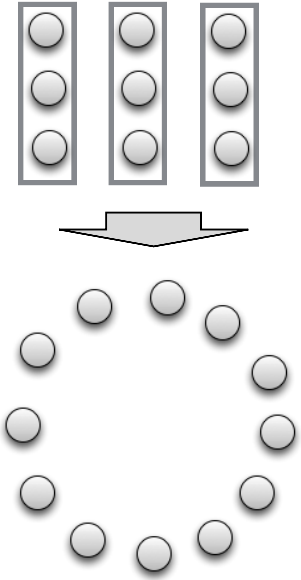
Ouch! Cost 1 for each request.

Lower bound of Ω(k) for competitive ratio, despite big augmentation!

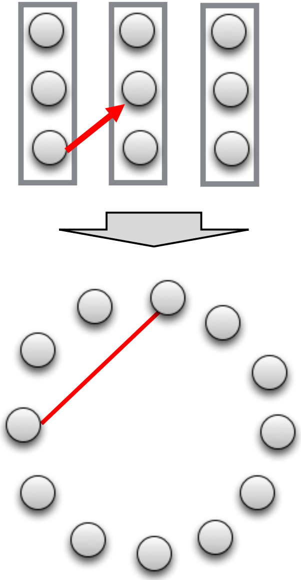
A Simple $O(n^2)$ Upper Bound

Algorithm

- Based on «growing communication components»
- Cycles through phases
    - Initially in each phase: empty graph of n nodes



A Simple $O(n^2)$ Upper Bound



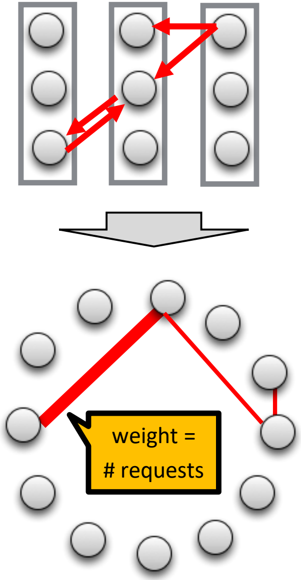
Algorithm

- Based on «growing communication components»
- Cycles through phases
    - Initially in each phase: empty graph of n nodes
    - For each inter-cluster request for ON: insert edge

A Simple $O(n^2)$ Upper Bound

Algorithm

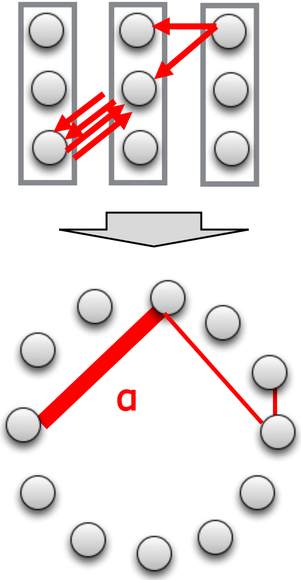
- Based on «growing communication components»
- Cycles through phases
  - Initially in each phase: empty graph of n nodes
  - For each inter-cluster request for ON: insert edge
  - Induces a «communication component»: edge weight = # requests



A Simple O(n^2) Upper Bound

Algorithm

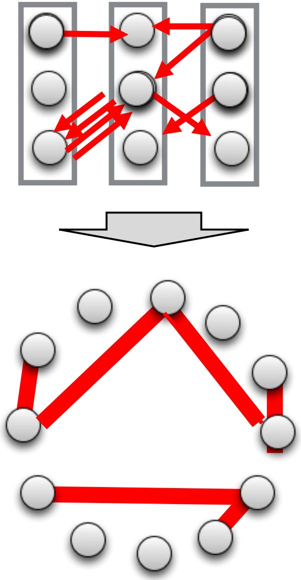
- Based on «growing communication components»
- Cycles through phases
    - Initially in each phase: empty graph of n nodes
    - For each inter-cluster request for ON: insert edge
    - Induces a «communication component»: edge weight = # requests
    - If an edge (u,v) weight reaches α, DET repartitions nodes, so that *all edges which have reached α so far are in same cluster*!



A Simple $O(n^2)$ Upper Bound

Algorithm

- Based on «growing communication components»
- Cycles through phases
  - Initially in each phase: empty graph of n nodes
  - For each inter-cluster request for ON: insert edge
  - Induces a «communication component»: edge weight = # requests
  - If an edge (u,v) weight reaches α, DET repartitions nodes, so that *all edge* *have reached α so far are in same*
  - If this is not possible: phase ends

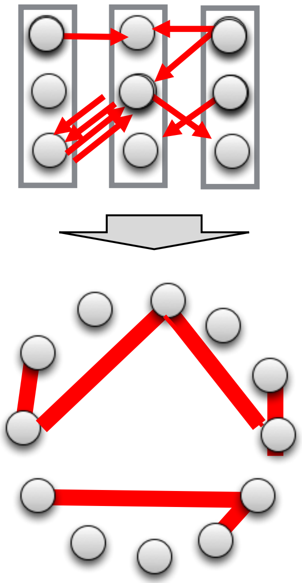


Components cannot be partitioned perfectly (first component alone too large)!

A Simple O(n^2) Upper Bound

Algorithm

- Based on «growing communication components»
- Cycles through phases
  - Initially in each phase: empty graph of n nodes
  - For each inter-cluster request for ON: insert edge
  - Induces a «communication component»: edge weight = # requests
  - If an edge (u,v) weight reaches α, DET repartitions nodes, so that *all edge* *have reached α so far are in same*
  - If this is not possible: phase ends



Components cannot be partitioned perfectly (first component alone too large)!

Competitive ratio?

A Simple O(n^2) Upper Bound

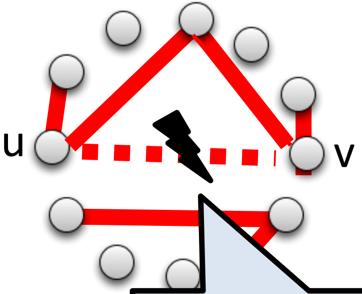
Analysis (costs per phase):

- Observe: edge weights always $\leq$ ɑ: once reach ɑ, their endpoints will always be collocated (by algorithm definition)

A Simple O(n^2) Upper Bound

Analysis (costs per phase):

- Observe: edge weights always ≤ α: once reach α, their endpoints will always be collocated (by algorithm definition)
- α-edges form a forest (so at most n many!): once two nodes (u,v) are connected by a path of α–edges, they are in a single cluster and will no longer communicate across clusters



Impossible: u and v already collocated by definition!

A Simple O(n^2) Upper Bound

Analysis (costs per phase):

- Observe: edge weights always ≤ α: once reach α, their endpoints will always be collocated (by algorithm definition)
- α-edges form a forest (so at most n many!): once two nodes (u,v) are connected by a path of α–edges, they are in a single cluster and will no longer communicate across clusters
- Thus: ON cost per phase:
  - At most 1 reorganization per α-edge (at most n α-edges), so n times reconfig cost n·α, so n^2α
  - Communication cost: at most α per edge (at most n^2 many), so also at most n^2α

A Simple $O(n^2)$ Upper Bound

Analysis (costs per phase):

- Observe: edge weights always ≤ α: once reach α, their endpoints will always be collocated (by algorithm definition)
- α-edges form a forest (so at most n many!): once two nodes (u,v) are connected by a path of α–edges, they are in a single cluster and will no longer communicate across clusters
- Thus: ON cost per phase:
    - At most 1 reorganization per α-edge (at most n α-edges), so n times reconfig cost $n \cdot \alpha$, so $n^2\alpha$
    - Communication cost: at most α per edge (at most n^2 many), so also at most $n^2\alpha$
- Costs of OFF per phase:
    - If OFF migrates any node, it pays at least α
    - If not, it pays communication cost at least α: the grown components do not fit clusters (intra-cluster edges only): definition of «end-of-phase»!

Upper bound of $O(n^2\alpha/\alpha)=O(n^2)$ for competitive ratio!

Known Results So Far

- Case k=2 („online rematching“): constant competitive ratio

- General case: with a little bit of augmentation: O(k log k) possible
    - Recall Ω(k) lower bound
    - Nice: independent of number of clusters!
    - Practically relevant: # VM slots per server usually small

Conclusion

- Communication networks become more flexible:
  - **Software**-defined: bring your own algorithm
  - **Topology** subject to optimization
  - **Placement** subject to optimization

- Challenges:
  - Consistent **reconfiguration**
  - (Dynamic) **network design**
  - Online **migration**

Thank You !

References

Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks
Stefan Schmid and Jiri Srba. 37th IEEE Conference on Computer Communications (**INFOCOM**), Honolulu, Hawaii, USA, April 2018.

Congestion-Free Rerouting of Flows on DAGs
Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht. ArXiv Technical Report, November 2016.

Taking Control of SDN-based Cloud Systems via the Data Plane
Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid.
ACM Symposium on SDN Research (**SOSR**), Los Angeles, California, USA, March 2018.

Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid. 31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

Online Balanced Repartitioning
Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. 30th International Symposium on Distributed Computing (**DISC**), Paris, France, September 2016.

SplayNet: Towards Locally Self-Adjusting Networks
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016.

What about  ?

What about  ?

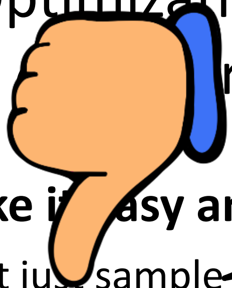
- Recall: weak adversary cannot choose request sequence but only the distribution
  - Adversary needs to sample i.i.d. from this distribution
  - Moreover: Adversary knows (deterministic or randomized) «learning» algorithm, i.e., chooses worst distribution

Any ideas?

The Crux: *Joint* Optimization of Efficient Learning *and* Searching

- **Naive idea 1: Take it easy and first learn distribution**
  - Do not move but just sample requests in the beginning: until exact distribution has been learned whp
  - Then move to the best location for good

The Crux: *Joint* Optimizatio
ro



Waiting can be very costly: maybe start configuration is very bad and others similarly good: takes long to learn, not competitive! Need to move early on, away from bad locations!

- **Naive idea 1: Take i asy and**
  - Do not move but ju sample requests in the beginning: until exact distribution has been learned whp
  - Then move to the best location for good

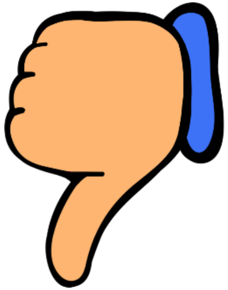
The Crux: *Joint* Optimization of Efficient Learning *and* Searching

- **Naive idea 1: Take it easy and first learn distribution**
  - Do not move but just sample requests in the beginning: until exact distribution has been learned whp
  - Then move to the best location for good

- **Naive idea 2: Pro-actively always move to the lowest cost configuration seen so far**

The Crux: *Joint* Optimization of Efficient Learning *and* Searching

- **Naive idea 1: Take it easy and first learn distribution**
  - Do not move but just sample requests in the beginning: until exact distribution has been learned whp
  - Then move to the best location for good
- **Naive idea 2: Pro-actively always move to the lowest cost configuration see**



Bad: if requests are uniform at random, you should not move at all! Migration costs cannot be amortized. Crucial difference to classic distribution learning problems: guessing costs!

The Crux: *Joint* Optimization of Efficient Learning *and* Searching

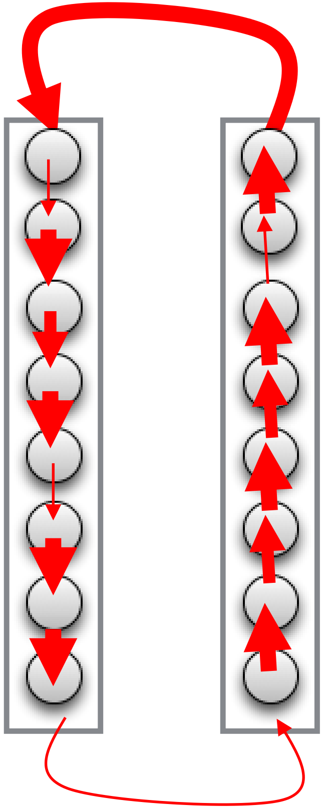
- **Naive idea 1: Take it easy and first learn distribution**
  - Do not move but just sample reques the beginning: until exac ribution ha learn o
  - Then move

Only move when it pays off! But e.g., how to differentiate between uniform and „almost uniform“ distribution?

- **Naive id st cost configurat**
  - Bad, e.g., if requests a stribute niformly at random: better not to move at all (mo ng costs cannot be amortized)

Example Learning Algorithm for Ring: Rotate Locally!

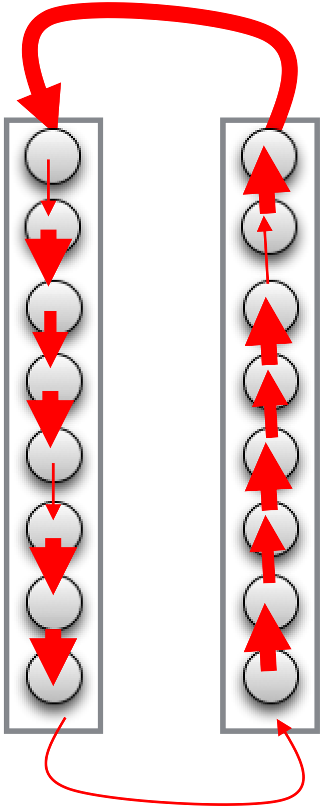
- ❑ Mantra of our algorithm: Rotate!
  - ❑ Rotate early, but not too early!
  - ❑ And: rotate locally



Example Learning Algorithm for Ring: Rotate Locally!

Define **conditions** for configurations: if met, **never go back** to it (we can afford it w.h.p.: seen enough samples)

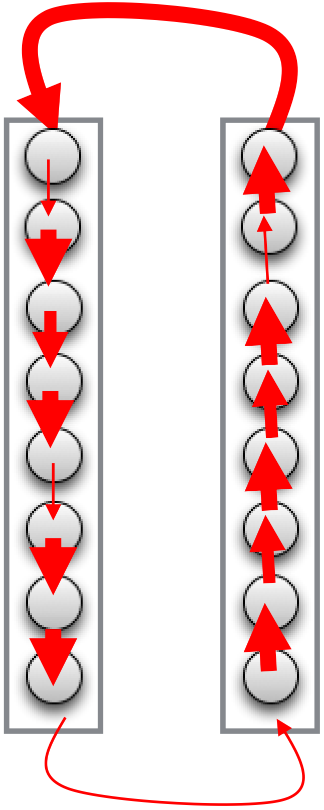
- Mantra of gorithm: Rotate!
  - Rotate early, but not too early!
  - And: rotate locally



Example Learning Algorithm for Ring: Rotate Locally!

- ❑ Mantra of our algorithm: Rotate!
  - ❑ Rotate early, but not too early!
  - ❑ And: rotate locally

If current configuration is **eliminated**, go to **nearby configuration** (in directed manner: no frequent back and forth)!

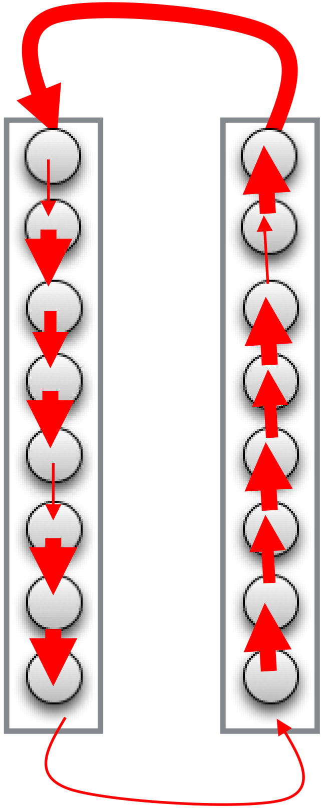


Example Learning Algorithm for Ring: Rotate Locally!

- ❑ Mantra of our algorithm: Rotate!
  - ❑ Rotate early, but not too early!
  - ❑ And: rotate locally

If current configuration is **eliminated**, go to **nearby configuration** (in directed manner: no frequent back and forth)!

Growing radius strategy: allow to move further only once amortized!



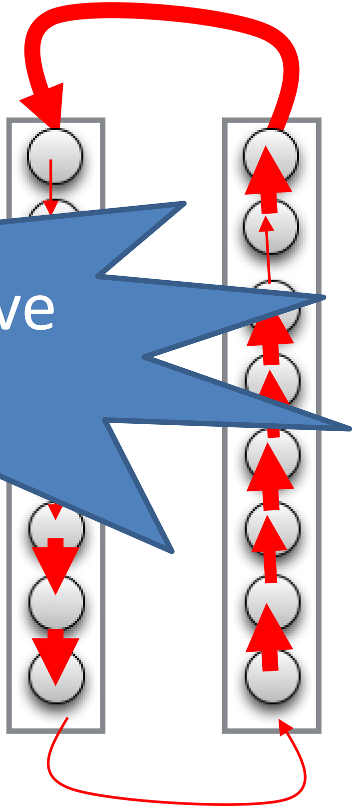
Example Learning Algorithm for Ring: Rotate Locally!

- Mantra of our algorithm: Rotate!
  - Ro te but n ea
  - An

log(n)-competitive w.h.p.

If current co guratio is eliminated, go to nearby configuration (in directed manner: no frequent back and forth)!

mo ier c onc ortized!



Future work

- More general graphs: regular/maximum degree $\mathrm{n}^{1/\mathrm{r}}$, for any r.
- Do we require alternate flavours of graph entropy?
- Maintaining the bounded degree network dynamically.

Further Reading

Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

rDAN: Toward Robust Demand-Aware Network Designs
Chen Avin, Alexandr Hercules, Andreas Loukas, and Stefan Schmid.
Information Processing Letters (**IPL**), Elsevier, 2018.

SplayNet: Towards Locally Self-Adjusting Networks
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.
IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016.